Center for Advanced Computation

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN
URBANA, ILLINOIS 61801

CAC Document Number 230
CCTC-WAD Document Number 7511

Networking Research in Front Ending
and Intelligent Terminals

**Offloading ARPANET Protocols
To A Front End**

September 30, 1977

CAC Document Number 230
CCTC-WAD Document Number 7511

Networking Research in Front Ending
and Intelligent Terminals

OFFLOADING ARPANET PROTOCOLS
TO A FRONT END

by

John D. Day

Prepared for the
Command and Control Technical Center
WWMCCS ADP Directorate
Defense Communications Agency
Washington, D.C. 20305

under contract
DCA100-76-C-0088

Center for Advanced Computatio
University of Illinois at Urbana-Cha
Urbana, Illinois 61801

September 30, 1977

Approved for Release: Peter A. A
Peter A. Alsberg, Princi

Table of Contents

Table of Contents
(continued)

Summary

Background

Under contract DCA100-76-C-0088, the Center For Advanced Computation of the University of Illinois at Urbana-Champaign is investigating the capabilities of network front ends. As a part of this contract, an experimental network front end (ENFE) is being developed to interface a WWMCCS H6000 to the ARPA Network and to conduct experiments with a host-to-front-end protocol (HFP). In this project, the HFP and associated higher level process-to-service protocols are used in offloading network access and Telnet facilities from the host to the front end. The ENFE necessarily implements only one of many possible ways to offload Telnet facilities. There are also other network facilities - particularly file transfer - which one might like to offload. In this document we present a broad survey of offloading strategies for the most important ARPA Network facilities. This survey should be useful as a basis both for the future design of expanded front ends and for quantitative studies to determine optimal offloading strategies.

Offloading the Telnet Protocol

The ARPANET Telnet protocol provides a standard method for interfacing terminals and terminal-oriented processes to each other. Three key ideas went into the development of the Telnet protocol:

1. the concept of a network virtual terminal (NVT),
2. the principle of negotiated options, and
3. a symmetrical view of terminals and processes.

A Telnet implementation must handle the translation between real terminal data representations and the NVT representation, the negotiation of options, and the opening and closing of network connections. Connection

handling can best be handled in the front end. Translation can be done in either the host or the front end. The best place to handle any particular option depends on the characteristics of the option. In this document, we discuss in detail the trade-offs involved in different degrees of offloading and provide a brief analysis of the potential for offloading each of the Telnet options. We also discuss which process-to-service protocols are needed to implement the various schemes. The symmetry of this protocol allowed us to develop a new process-to-service protocol (the network virtual terminal PSP) designed to efficiently implement an intermediate level of Telnet offloading. (The maximum offloading strategy adopted for the ENFE was implemented with two separate PSP's - one for the user side and one for the server side.)

Offloading the File Transfer Protocol

The ARPANET File Transfer Protocol (FTP) consists of two major sections: a command-and-response section and a data transfer section. The File Transfer Protocol is inherently asymmetric. Commands are sent from the User FTP to the Server FTP to provide for user authentication, to specify parameters for data transfer, and to request service. We have identified two major aspects of FTP that are candidates for offloading: the data transfer process and the bookkeeping and marker handling required for restarting a transfer that has aborted. Since FTP makes use of the Telnet protocol, the Telnet functions can also be offloaded. We found that for User FTP there are eight possible offloading schemes that differ from one another in important ways.

The offloading of Server FTP presents an even more complicated problem. File systems in the host must be distinguished from file systems in the front end. For another thing, Server FTP is not the simple sequence of stages which characterizes User FTP. (An exception seems to be the data translating and reformatting functions, which we

believe should generally be assigned to the front end.) We have therefore confined ourselves to examining the individual FTP commands, classifying them as to whether they must be handled in the host or can be handled in the front end. In some cases (e.g., user authentication) a command is likely to be handled by both the front end and the host.

For offloading FTP, we have designed three new process-to-service protocols: a User FTP PSP, a Server FTP PSP, and a File Access PSP. The last provides a general facility for transferring files between host and front end. This facility should also be useful for purposes other than offloading FTP.

<u>Offloading Other ARPANET Protocols</u>

Although our major contractual obligation was to study the offloading of FTP (and Telnet as a natural conjunct of this), we also looked briefly at three other protocols: Remote Job Entry (RJE), Teleconferencing, and Network Graphics.

RJE makes heavy use of FTP; thus in studying FTP we have covered the major points of interest in offloading RJE. There are a few functions, of course, specific to RJE itself; these in general can not be offloaded.

There is no ARPANET Teleconferencing Protocol as such. However, we felt that teleconferencing is a sufficiently important network service to be included in this study. Our conclusion is that teleconferencing can and should be entirely offloaded, provided that the front end has its own file system. If the file system of the host must be used, all of the teleconferencing functions should still be placed in the front end, with the File Access PSP used to read and write files when necessary.

We also find the Network Graphics Protocol highly suitable for offloading. An exact assignment of duties, however, would depend heavily on the capabilities of the graphics terminal, the host and the front end.

Introduction

Network software is a significant burden on large hosts connected to a resource sharing network. This burden might be lessened through the use of a network front end. A network front end is a minicomputer interposed between host and network. Some network software functions could be "offloaded" to the network front end. This study explores what network software functions could be offloaded to a network front end. The Host-to-Front-End Protocol (HFP) [19] serves as the vehicle for this study. Familiarity with HFP is assumed.

Connecting a large computer to a packet-switched network requires a rather large investment in the development or modification of network software. The host-to-network interface or protocol must be implemented. An end-to-end protocol is required for general interprocess communication. These programs are complex and, if the operating system is unsuited to networking, may be large. Modification of the operating system is often necessary. Moreover, in order to make network services available to the user a host must implement one or more higher level protocols. Examples of higher level protocols are virtual terminal protocols, file transfer protocols, remote job entry service, graphics protocols, etc. But the network software burden on a host is not only in large one-time costs for software development, but also in large recurring costs for the processor, memory, and channel time consumed by the network software, and in software maintenance. It has been hoped that most of this burden could be offloaded to a relatively inexpensive network front end. A front end could also transform network communications to make them more suitable to the host. Standardization of the front end would allow the same

system to be used to front end a variety of hosts. This would save development and modification costs. On the other hand, tailoring the front end to a specific host would improve performance. There is clearly a tradeoff between these two goals.

Host-to-Front-End Protocol

The vehicle for the present study is the Host-to-Front-End Protocol (HFP) [19]. This protocol can be divided into three major sections:

1. a link protocol for control and communication on the actual physical connection between the two machines,
2. a channel protocol that provides a set of error-free, reliable, logical channels between processes in the host and services in the front end, and
3. a family of process-to-service protocols (PSP's) that provide the means for offloading the network protocols.

The channel protocol defines a set of Command and Response Messages. Each Message has a header field containing control information and a text field containing data. Four pairs of Commands and Responses provide communication between the host process and the front-end service. The PSP's use the text fields of the channel Commands and Responses to send PSP commands and responses between the process and the service. The channel Commands and Responses are:

The BEGIN Command and Response are used to establish a logical channel between the host and the front end.

The END Command and Response are used to terminate a logical channel.

The TRANSMIT Command and Response are used to transfer data.

The EXECUTE Command and Response are used to send requests out-of-band with the TRANSMIT Commands and Responses.

Each PSP is defined in terms of the channel protocol. A PSP defines how the text field of each channel protocol Command and Response is used for a particular service.

A PSP represents a class of offloading strategies for one or more protocols. The six PSP's are:

1. The Program Access PSP, which provides a general facility by which a process in the host may invoke a terminal-oriented program or system command in the front end.
2. The File Access PSP, which provides a general facility for moving bulk data between the host and the front end. It is used in offloading FTP or similar protocols.
3. The ARPA Host-Host PSP, which provides access to the ARPANET Host-Host protocol implementation in the front end.
4. The Network Virtual Terminal PSP, which provides low-level process-oriented classes of offloading of the Telnet protocol.
5. The User FTP PSP, which provides low-level process-oriented classes of offloading of the user side of the File Transfer Protocol.
6. The Server FTP PSP, which provides classes of offloading for the server side of a File Transfer Protocol.

Future Work

While this study explores what could be offloaded, it does not determine what should be offloaded. A given installation will attempt to provide a particular class of service to its users; for example, a timesharing service, a data management service or an archival service. The installation may have to handle a large number of terminals or perhaps only a few. Because of the finite resources of the front end one may not be able to offload everything and still provide adequate performance for the preferred services. For example, a

timesharing service supporting a large number of terminals and not much bulk transfer might maximize the offloading of Telnet and minimize response time. On the other hand, an archival service or a data management service might maximize the offloading of FTP and minimize the offloading of Telnet. The nature of these tradeoffs must be investigated. A methodology for determining how an installation should configure its front end must be developed. Similar work has been done for satellite graphics processors [18, 34, 35]. This work might be extended and adapted to the offloading problem. Also, the modeling framework [16] recently developed at CAC could be used to aid the investigation of these questions.

General Characteristics of Offloading

It is possible to discern a pattern in the way protocols could be offloaded. This pattern is related to the symmetry of the network protocol. A symmetrical protocol is one whose operation is identical for both sides. The ARPA Host-Host and Telnet protocols are symmetrical. In the Host-Host protocol no distinction is made between source and destination or between user and server. Either side may initiate the connection. The passing of data from one host to another is treated identically regardless of direction. In Telnet no distinction is made between terminals and processes. On the other hand, the FTP protocol is not symmetrical. One side (User) must initiate the connection, generate commands and receive responses; while the other side (Server) receives commands, performs the indicated task and sends responses.

As one would expect, symmetrical protocols offload in one way and asymmetrical protocols offload in another. For a symmetrical protocol, a single PSP is used for offloading; for an asymmetrical protocol two PSP's are required - one for each side.

Schemes for offloading the user side of an asymmetrical protocol show one pattern and those for offloading the server side show a distinctly different pattern.

This is a consequence of the structural differences between the two sides. The user side is simply a sequence of stages. A message from the user is transformed successively by each stage, e.g., the user interface, the protocol interpreter, the Telnet interface, and the NCP. But on the server side, the protocol interpreter must perform host-specific operations. On the server side, these host-specific operations

are the crux of the matter. While the sever side has stages analogous to those of the user side, they are of less importance, particularly with respect to offloading.

As a general principle, the structure of the user side of an asymmetrical protocol favors offloading in terms of stages. Thus, for the user side, there are the following offloading strategies:

1) Offload everything, leaving a simple process in the host which relays data between the user and the front end;
2) Leave the user interface in the host;
3) Offload nothing at this protocol level.

There may be variations on these three strategies, but as major classes they always appear.

The structure of the server side of an asymmetrical protocol favors offloading in terms of the operations. Most of these operations deal with the handling of protocol commands. (See figure 1.) Some operations <u>must</u> be done in the host, some <u>must</u> be done in the front end. Some can be done either place. Thus the question of offloading strategy is reduced to those cases for which there is a choice. It is then a question of the relative advantage or disadvantage of having each such operation performed in the host or in the front end. Since offloading an operation to the front end may not only affect the performance of the front end in undesirable ways, but may also require more complex mechanisms to control the offloaded operation from the host, the net gain to be realized by such a strategy must be carefully determined. Large variations in operating system architectures make it difficult to determine which operations should be offloaded without careful investigation of individual cases.

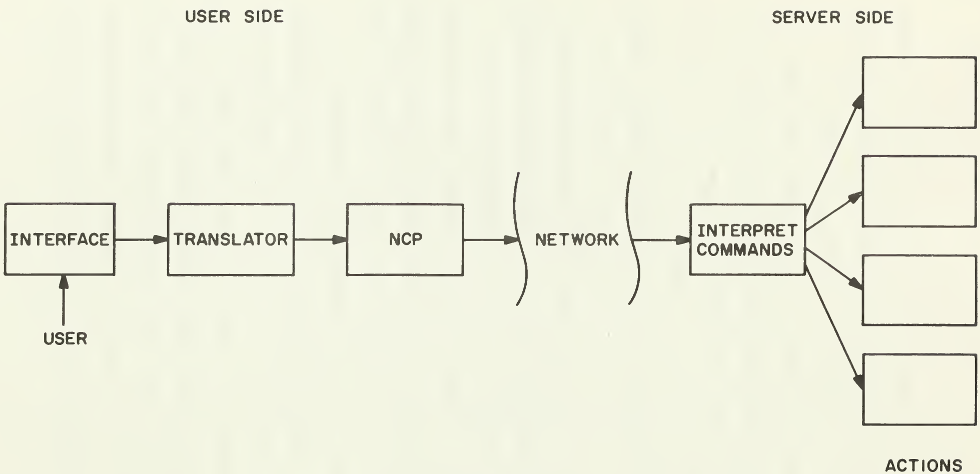


Figure 1

Flow of Commands in an Asymmetric Protocol

PSP's may also differ by being location-dependent or location-independent. Most are location-dependent. The process or user on whose behalf a service is performed is located in the host and the service is in the front end. However, there are some PSP's, such as the File Access PSP, which make the distinction between a user and a service but do not require their association with the host or front end.

Two General-Purpose Process-to-Service Protocols

Two of the PSP's used in this work to offload network software are not associated with a particular network protocol. One is the Program Access PSP and the other is the File Access PSP.

The Program Access PSP (see appendix 1) is a general purpose protocol that combines some of the functions of a virtual terminal protocol with a simple mechanism allowing host processes or users to invoke programs or commands in the front end. The Program Access PSP (PA-PSP) can be used to provide a direct connection from the user's terminal to a front-end program with minimal interference by the host. Depending on the nature of the terminals and the operating system in the host, the process side of a PA-PSP may have to carry out such operations as:

querying the user for usercode and password,

handling break characters from the terminal,

flushing data going to the terminal,

handling half-duplex terminals,

diverting binary or character data to auxiliary devices, and

modifying terminal parameters that control echoing.

Of course, the more items from this list that are required, the less the savings to be gained by using this PSP. Also, since this PSP may communicate with the front end system at the level of the user command language interface, it will be inappropriate for use by processes.

The File Access PSP (see appendix 2) is a general purpose protocol that facilitates the movement of files or bulk data among the host, the front end and the network. It allows the host to access the front-end's file system or vice versa, and the host to transfer data

between its file system and the network. The File Access PSP (FA-PSP) allows the user to access any point in a file and supports a mechanism for restarting file transfers. This mechanism is compatible with most File Transfer Protocols.

Offloading the Telnet Protocol

The ARPANET Telnet protocol provides a standard method for interfacing terminals and terminal-oriented processes to each other. The protocol is based upon three ideas:

1. the concept of a Network Virtual Terminal (NVT),
2. the principle of negotiated options, and
3. a symmetrical view of terminals and processes.

When a Telnet connection is established, each end is assumed to terminate in an NVT. The NVT represents a canonical terminal. It was designed to strike a balance between being overly restrictive and overly inclusive. The principle of negotiated options provides the mechanism by which terminals and processes that find the NVT too restrictive may enhance its properties [23]. Such options include changing the echo mode, the line width, or the page size, and setting tabstops. For a more complete description of the protocol and the current options, the reader should consult references [4-13,17,20,23-31,36,37].

In most systems there are two major uses of the Telnet protocol. First, it is used as a general terminal-oriented communication medium. Second, a host may have a local interactive program (User Telnet) that uses the Telnet protocol to connect a local user to a remote host as if that user were directly connected to the remote host. Those aspects of the protocol itself that may be offloaded are character translation, connection manipulation, and some of the options. The User Telnet application program in a user host may offload all of the above functions and the user interface to the User Telnet program. The section below takes a closer look at how the offloading may be done.

Offloading the User Telnet Application

There are three basic schemes for offloading the User Telnet Application: maximum offloading, placing the user interface in the host, and placing everything in the host. (See table 1.)

Scheme Number	Functions in the Host	Functions in the Front End	PSP's Used
1	Local terminal handling Relay Process Go-Aheads	User Interface Telnet Connection handling	PA-PSP
2	User Interface Local Terminal Handling Relay Process Go-Aheads	Telnet Connection handling	NVT-PSP
3	User Interface Telnet Local Terminal Handling Relay Process Go-Aheads	Connection Handling	ARPA-HH

Table 1. User Telnet Offloading Schemes

For the maximum offloading scheme (see figure 2), the Program Access PSP is used. As noted above, this scheme requires a "relay" process in the host to move data between the user's terminal and the program access service (PAS) module in the front end. There are two disadvantages to this scheme. Firstly, as noted above, the more functions required of the relay process, the more complex the relay process must be and the less that might be saved. In fact, the relay process may become so complex as to require a user interface for requesting information from the user. This is exactly the part of Telnet that the PA-PSP was intended to offload. Secondly, processes in the host that wish to use Telnet must use a human-oriented interface. This may be unwieldy and overly complex, and in some cases impossible. In these cases the Network Virtual Terminal PSP (NVT-PSP) should be used.

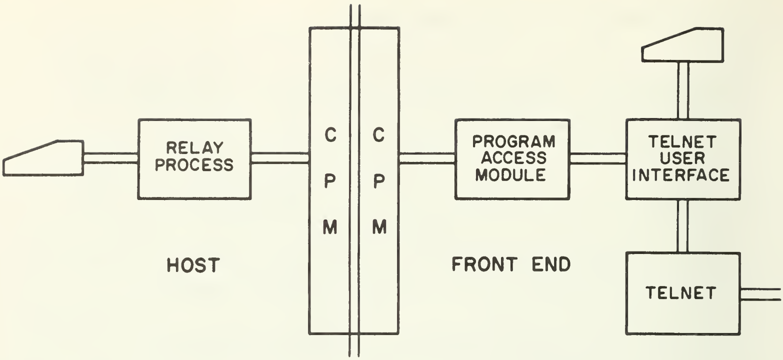


Figure 2

Maximum Offloading of User Telnet

In order to support some Telnet options, the relay process may have to map them into local terminal parameters. The best example of such a case is a system that wishes to support the remote controlled transmission and echoing (RCTE) option. When the option is negotiated, the terminal handler in the host must be notified of the change in echoing and transmission. It may not be possible to handle this case at all with the PA-PSP scheme, but the NVT-PSP is capable of handling it easily.

The Network Virtual Terminal PSP (see appendix 4) provides the facilities for the second offloading scheme shown in table 1. The user interface is placed in the host and uses the NVT-PSP to access the User Telnet process in the front end. (See figure 3.) The only problem created by this scheme is that two possibly incompatible user interfaces will be required, one in the front end and one in the host. The advantage of this scheme is that it provides a process-oriented interface to the

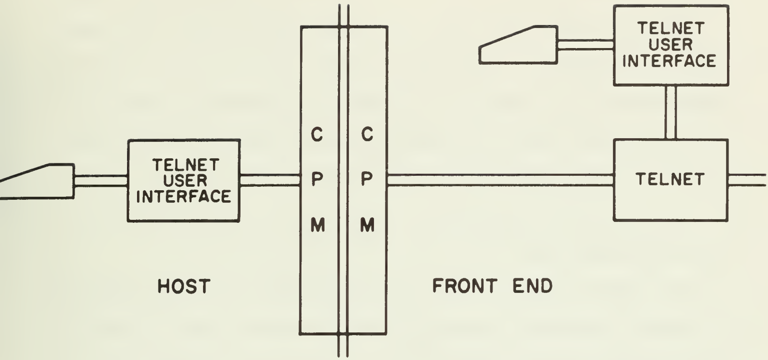


Figure 3

Telnet User Interface in the Host

host side. Thus programs don't have to decipher responses intended for a human. Some systems may use both the maximum offloading scheme (for the high use User Telnet application) and the NVT-PSP for use by resource sharing programs. Also, systems that require a rather complex relay process to implement the PA-PSP are perhaps better off using just the NVT-PSP. Since the NVT-PSP and the Telnet protocol are both symmetrical, the details of using this scheme to offload the User Telnet application are identical to the offloading of the Telnet protocol itself.

Offloading the Telnet Protocol Implementation

The functions of the Telnet implementation are manipulating network connections, negotiating Telnet options, mapping between local and network representations, transmitting data over connections, handling special control functions, and interfacing remote terminals so that they appear as local terminals. (Since the protocol is symmetrical, this is true of both the User and Server sides). The offloading scheme

represented by this model places connection manipulating and data transmitting in the front end. The interfacing of remote terminals so that they appear as local terminals must be done in the host. The scheme is flexible in that the remaining functions (option negotiation, special control functions, and translation between local and network representations) may be implemented in either the front end or the host, depending on local installation constraints. Typically the translation will be done in the front end.

There are several Telnet control functions that one would like to consider offloading. These are Interrupt Process, Break, Are You There and Abort Output. Since for all of these except the last the effect of the control function is on the process itself, the most that can be done is to ensure that the control function arrives at the earliest possible moment. This is done by sending it to the host via the out-of-band EXECUTE Command.

In the case of Abort Output a little more can be done. When the NVT service in the front end receives an Abort Output control function, it should pass the control function to the host via the EXECUTE Command. Since this HFP Command is sent out-of-band with respect to the data, the Abort Output will be sent to the host as early as possible. The front end can then respond to the Abort Output control function according to the Telnet protocol by sending a Host-Host Protocol INS command associated with this connection to the remote host, flushing any data it has buffered for transmission to the network and continuing to flush data until a Telnet Data Mark is received from the host. The front end may send a Telnet Data Mark to the remote system as soon as it has flushed all queued data or it may wait for the Data Mark from the host. (If it does the former, it should not pass the Data Mark received

from the host on to the remote system.) When the host receives the Abort Output control function, it should flush all data it has queued for the front end, and send a Telnet Data Mark to the front end. (Note: Neither side should request the HFP Channel Protocol to flush data. The Telnet protocol states that an Abort Output flushes only data, not control functions, and a channel-level flush does not distinguish between the two.)

An installation may take either of two basic approaches to offloading the Telnet Options. It may offload nothing, in which case the Telnet service in the front end need only reformat the data stream, perhaps do character translation, and handle the network connections. Or, the installation may wish to offload as much as possible (given its host's characteristics) in which case the front end must also be able to negotiate and execute some of the Telnet options.

Unfortunately, it is not possible to say definitively for some of the Telnet options whether they should be offloaded or not. Table 2 can be used as a guide to how easily the various options may be offloaded.

In most cases, two attributes of each option determine whether or not it can be offloaded:

1. An option may or may not affect the host's terminal-handling parameters.
2. An option's taking effect may or may not require synchronization with a particular point in the data stream.

If an option affects the host's terminal-handling parameters, then, although some of its processing may be offloaded, some processing will have to be done in the host.

If an option requires synchronization with the data stream, it may be very difficult, if not impossible, to offload.

Option Name	Telnet Option Num.	Host Dep.?	Coord w/Data Stream	Can be Offloaded?
Approximate Message Size	4	no	no	yes
Binary Transmission*	0	yes	yes	no
Byte Macro*	19	yes	yes	yes
Data Entry Terminal*	20	yes	yes	maybe
Echo*	1	no	yes	no
Extended ASCII	17	no	yes	yes
Logout*	18	yes	no	no
Output Carriage Ret. Dispos.	10	no	no	yes
Output Formfeed Dispos.	13	no	no	yes
Output Horz. Tab Stops	11	yes	no	no
Output Horz. Tab Stops Dispos.	12	no	no	yes
Output Linefeed Dispos.	16	no	no	yes
Output Line Width*	8	?	no	maybe
Output Page Size*	9	?	no	maybe
Output Vert. Tabstops	14	yes	no	no
Output Vert. Tabstops Dispos.	15	no	no	yes
Reconnection	2	no	yes	yes
RCTE*	7	yes	yes	no
Status*	5	yes	yes	some
Suppress Go Ahead*	3	no	no	yes
Timing Mark	6	yes	yes	no

Table 2. Classification of Telnet Options

*Offloading these options is discussed in further detail in the text.

Table 2 lists the current ARPANET Telnet options. Those options marked with an asterisk are discussed below in greater detail. For the rest, the table indicates whether they can be offloaded. Note that all options that are not supported by the host may be offloaded; i.e., the front end can refuse negotiations without involving the host.

Binary Transmission. In most cases, this option cannot be offloaded. However, the fact that it has been negotiated may be of interest to the service module in the front end. If the front end has been doing (or normally does) some translation tasks for the host, it will be necessary to coordinate the beginning of binary data with the cessation of such translation.

Byte Macro. If this option is supported and any other options or functions are offloaded, then this option must be offloaded. This option provides a means for mapping arbitrary strings into one byte. Therefore, when this option is in effect, the macro expansion must take place at the earliest point.

Data Entry Terminal (DET). It may be possible to offload all or part of this option depending on the nature of the host system. If the host supports only one form of data entry terminal, then the front end can map many DET option subcommands into the form required by the local terminals or programs.

Echo. In general, it is not possible to offload this option, since the process in the host which is connected to this terminal may wish to echo characters other than those typed. However, there are cases where it might be desirable if the front end did handle this option for certain kinds of hosts.

For example, consider computers that support only half-duplex terminals, such as IBM 360's, Burroughs 6700's, or Honeywell 6000's.

Suppose that a user connects to such a host and asks it to DO ECHO. It would be quite reasonable to have the front end handle the echoing and present the host with an apparent line-at-a-time environment. This would avoid many of the difficulties of trying to fit a line-at-a-time system into a character-at-a-time mode.

Output Line Width. Once again the decision to offload this option rests heavily on how line width regulation is enforced. If the regulation of line width is done by simple "line folding", then this function may be performed by the front end. However, if regulating line width requires more fundamental adjustments or more sophisticated action, it is quite likely that it will not be possible to offload this option.

Output Page Size. The decisions that are relevant for offloading this option are similar to the ones described for output line width. Some systems may interpret the regulation of page size as follows. The maximum number of lines in a page is sent. If there is still data to be sent, the sender pauses, waiting for an input from the user indicating that he is ready to proceed. If this is the sort of regulation desired when this option is negotiated, then it can be offloaded. However, if more fundamental adjustments or more sophisticated action is required, it may not be possible to offload it.

Remote Controlled Transmission and Echoing. The RCTE option is meant as a more general mechanism to replace the Echo option. If RCTE is used to replace the Echo option, then it may be offloaded under the same conditions as the Echo option. Otherwise, offloading is not possible and actually defeats the purpose of the RCTE option.

Status. Since not all options may be offloaded, it is necessary to offload part of the Status option and to leave part of it in the front end. It is suggested that the Status option be handled in the following way:

When the service module in the front end receives a Status option negotiation, it will respond favorably to the request by sending a WILL STATUS to the sender. It will then relay the option to the host in the normal manner and wait for a response from the host. When the host receives the DO STATUS command from the front end, it will transmit back to the front end the subnegotiation specifying the state of the options it handles. When this subnegotiation is received by the front end, it will insert into this subnegotiation the status of the options it handles and then it will send the completed subnegotiation to the foreign host.

If no options are supported in the host or if no options are supported in the front end, then, of course, this procedure may be avoided and the whole operation done in the front end or in the host, respectively. Also, it is possible for the front end to know what options are not supported and supply the default status information.

Suppress-Go-Ahead. In a sense, the Suppress-Go-Ahead option is always offloadable. If the host always generates go-aheads, then it is no problem for the front end to filter them out when the option is negotiated. However, it is impossible for the front end to insert go-aheads.

Telnet "Synch" Sequences. In some cases, searching for "interesting" characters in the data stream may be offloaded. For example, if the only characters deemed "interesting" are the Telnet special characters, then this function may be offloaded. Or the front end may be provided with a list (on either a static or dynamic basis) of strings to be searched for. However, even if these functions are performed in the front end, they must also be done in the host before the "synch" sequence is received.) The only reason for offloading this function is to increase the response to the user's request.

The ARPANET File Transfer Protocol (FTP) [21,22] consists of two major sections: a command-and-response section called the Protocol Interpreter (PI) and a data transfer section called the Data Transfer Process (DTP). The model used in the FTP specification is shown in figure 4. The Telnet protocol is used to control the character-oriented command connection. Commands are sent from the User FTP module to the Server FTP module. These commands consist of a verb and single parameter. The commands may be divided into three major classes:

1. access control commands (USER, PASSWORD, etc.),
2. transfer parameter commands (TYPE, MODE, etc.), and
3. service commands (RETRIEVE, STORE, DELETE, etc.).

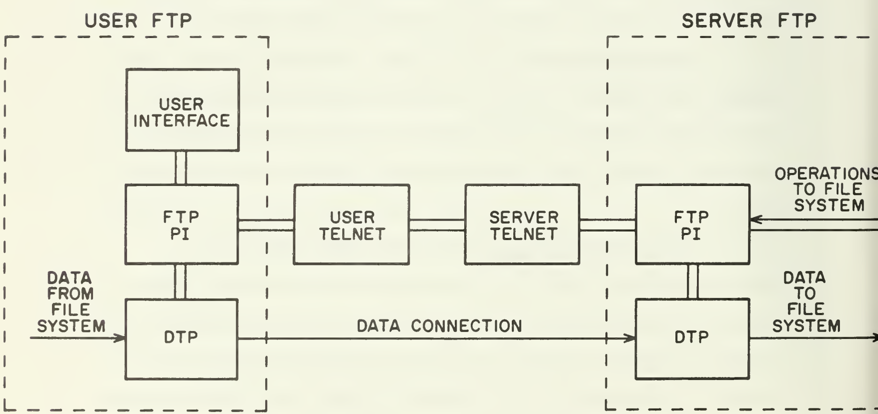


Figure 4

ARPANET FTP Model

The Server FTP module responds to each command with one or more FTP replies. A reply consists of a three digit code called a reply code and one or more lines of text. The reply code makes the replies useful to programs, while the text makes available more complete information to a human user.

The transfer parameter commands are used to coordinate the mechanism for transferring the data, and to describe the format of the data to be transferred. Once these parameters have been agreed on, the User and Server FTP's open a separate connection for transferring the data. This is loosely referred to as the data transfer process or DTP.

There are two major aspects of FTP that are candidates for offloading: the transfer restart mechanism and the data transfer process. Although restart does require direct manipulation of the file and this function cannot be offloaded, there is a fair amount of restart marker "bookkeeping" that must be done. The front end can keep track of the markers and the position they correspond to in the local file. The primary functions of the data transfer process are manipulating the data connection and doing any translation of the data into a form suitable for storage in the local file system or for sending to the remote site. Offloading this translation function also means that, in most cases, the transfer parameter commands that are used to define the translation may be offloaded as well. Offloading translation should provide considerable savings to hosts which transfer data to or from computers unlike themselves, especially if the front end has efficient bit-level operations.

The User and Server FTP offloading models described here use the File Access PSP to offload the data transfer portion of the FTP. We will note some of the properties of this File Access PSP as we describe

the models. We complete this section with a review of the requirements for the File Access Service.

For systems that don't allow access to FTP from terminals attached to the host, the File Access Service (FAS) module is the only portion of software for a User FTP that must be in the host.

If file systems are supported in both the host and the front end, then some way must be provided to distinguish a pathname as belonging to one or the other. Since remote filenames are the only ones sent as parameters in the FTP commands, an offloaded User FTP in a system with files in both the front end and the host only needs some local mechanism in the user interface for distinguishing the two file systems. Given this information as part of a command, the FTP user interface or protocol interpreter can access the correct open file. However, when an offloaded Server FTP which serves file systems in both the front end and the host is involved, some property of the pathname transferred in the data transfer command must be used to indicate where the file is. Several techniques might be used. The pathname might be prefixed with the characters HOST or FRONT END to indicate where the file is. The front end might have a partial copy of the host's file system directory indicating which files are where. Or, a non-technique might be used: attempt to access the pathname on the front end; if the attempt fails, try to access it on the host.

Offloading the User FTP

We have identified eight distinct schemes for offloading User FTP. These are summarized in Table 3 and described in detail in this section.

1. Maximum offloading. This scheme is analogous to the maximum User Telnet offloading scheme. As shown in figure 5, this model uses the Program Access Service (PAS) to provide a full duplex path to the User FTP in the front end. The File Access Service (FAS), which may

Scheme Number	Functions in the Host	Functions in the Front End	PSP's Used
1	Relay Process FAM	User Interface FTP-PI PTP-DTP Telnet Host-Host Restart	PA-PSP FA-PSP
2	FTP User Interface FAM	FTP-PI FTP-DTP Telnet Host-Host Restart	FTP-PSP FA-PSP
3	FTP User Interface FTP-PI FAM Restart	FTP-DTP Telnet Host-Host	UT-PSP FA-PSP
4	FTP User Interface FTP-PI FAM Telnet Restart	FTP-DTP Host-Host	ARPA-HH FA-PSP
5	FTP-DTP Relay Process	User Interface FTP-PI Telnet ARPA-HH Restart	PA-PSP FA-PSP
6	User Interface FTP-DTP	FTP-PI Telnet ARPA-HH Restart	FTP-PSP FA-PSP
7	FTP-DTP User Interface FTP-PI Restart	Telnet ARPA-HH	UT-PSP ARPA-HH
8	FTP-DTP User Interface FTP-PI Telnet	ARPA-HH	ARPA-HH

Table 3. User FTP Offloading Schemes

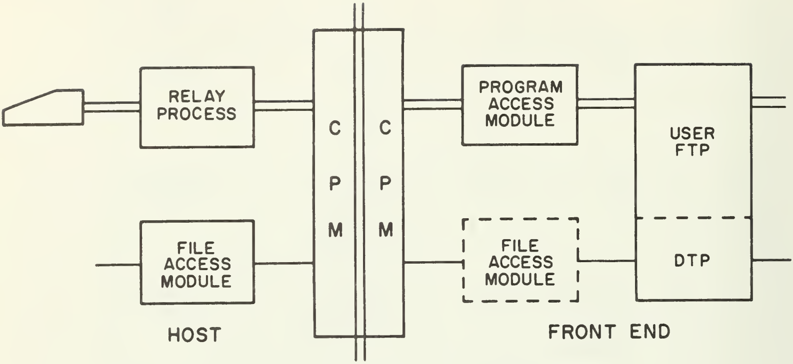


Figure 5

Maximum User FTP Offloading

be synonymous with the data transfer process of the FTP, is used to move the data to the proper file in the host. The FAS module must be able to log a user into the host, open a file, read or write to it, abort a transfer cleanly, handle restart reporting and positioning, return error conditions on access, etc.

With this scheme, the PAS in the host and the FAS in the host are completely independent. Any decision to transfer a file and thus use the FA-PSP is recognized only by the User FTP in the front end. Thus, use of the FA-PSP is initiated by the front end. There are several ways to handle this: 1) the host can always have a BEGIN Command outstanding for a connection to the FAS, or 2) the front end can send a BEGIN Command to the host requesting service. The first case is analogous to the mechanism used in the Network Virtual Terminal PSP. However, there is one major difference. The user of the FA-PSP must be logged in to the host in order to access the file. In order to minimize

the amount of negotiation required before a transfer can commence, the BEGIN Commands should probably go to the host.

The other major requirements for this scheme are that the FAS module must be able to report back to the User FTP in the front end that all records on bytes preceding a restart marker have been actually written onto the file, and that the service in the front end must be able to translate a marker into a local file position in the event a restart is required.

2. FTP user interface in the host. This scheme moves the FTP user interface into the host. (See figure 6.) Several advantages result from this configuration. First, since the user interface knows when the user has requested a transfer, it may initiate the FA-PSP. This strategy avoids the necessity of sending BEGIN Commands to the host. The FA-PSP would use the "channel" connection type (see appendix 3) to open the data connection on the default transfer sockets. For similar reasons the protection problem encountered in the above model (i.e., the user's having to log in to the host) is avoided. However, if the file is in the front end's file system, the user may have to log in to the front end. This would depend on the particular security and protection arrangements.

Restart bookkeeping can be done in either the host or the front end. If done in the host the FAS module in the host must be able to pass information to the user interface to inform the user of the markers. However, if the restart management is done in the front end, the same mechanisms described above apply.

If there is no file system in the front end, then the FTP-PI has very little to do except act as a relay for commands and responses and initiate the data transfer process (DTP); the user interface can

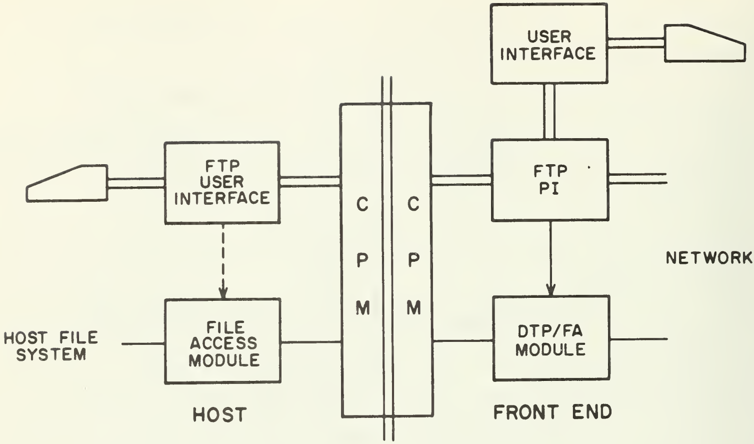


Figure 6

FTP User Interface in the Host

generate the actual FTP commands. If a file system does exist in the front end, then the user interface must be able to tell the DTP (or the DTP must be able to determine) in which file system the file to be accessed resides.

The major disadvantage to this scheme is that two user interfaces are required to the FTP-PI (one in the host and one in the front end), and there is every likelihood that they will not be the same.

The functions offloaded by this scheme are connection handling, User Telnet, protocol state management, the DTP, and possibly restart.

3. <u>Telnet and Data Transfer in the FE</u>. This scheme (see figure 7) primarily offloads the Telnet and data formatting functions. It is necessary that the FTP-PI be able to explicitly coordinate the data connection with the Telnet connection or always use the SOCK

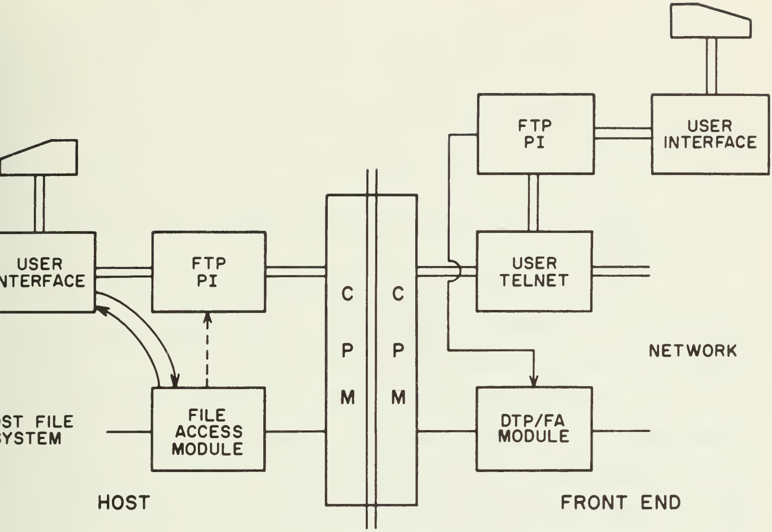


Figure 7

Telnet and Data Transfer in the FE

command. If the PA-PSP is used to access Telnet then the FTP-PI must be able to specify the contact socket to the Telnet user interface and it must also be able to manipulate the human-oriented Telnet interface. This is probably a good candidate for using the Network Virtual Terminal PSP.

There must be communication between the FTP-PI and the FAS in the host to specify the mode, type, etc., that will be used for the transfer. The most reasonable way to handle this is to pass the information through the FAS module in the host.

Although restart management could be handled in the front end, it seems to be overly complex to consider such an assignment. (The host

FAS module would have to tell the front-end FAS module that data had been written. Then the front end would have to pass that information to the user back through the FAS module.) Thus, it seems most reasonable to assume that restart would be done in the host for this model. Restart management in the host could be done without additional complexity in the FA-PSP. All that would be required would be for the restart report to be channelled to the FTP-PI in the host rather than to send it back to the front end.

This model requires two user interfaces and two FTP-PI's if access is allowed through the front end and the host. Problems might arise since these may be significantly different.

4. Data Transfer in the Front End. In essence, this model (see figure 8) merely offloads some of the connection manipulation and data translation and re-formatting. Again it is necessary that the FTP-PI in the host and FAS module in the front end be able to communicate. This would be done through the FAS module. It is also necessary for the FTP-PI to be able to coordinate its control connection with the data connection opened by the FAS or to use the FTP SOCK command. If a file system exists on the front end then it may be necessary for the FAS module in the host to log in to the front end on behalf of the user.

Data Transfer in the Host. There is a variation of each of the above offloading models that puts the data transfer process in the host. We will consider each one in turn:

5. Only DTP in the host. This scheme (see figure 9) has many of the same problems as its counterpart (scheme 1). Since the relay process is independent of the DTP, the front-end side of the FAS must log in to the host. Thus BEGIN's will generally go to the host. In this case, however, more information (mode, type, etc.) must be sent to the

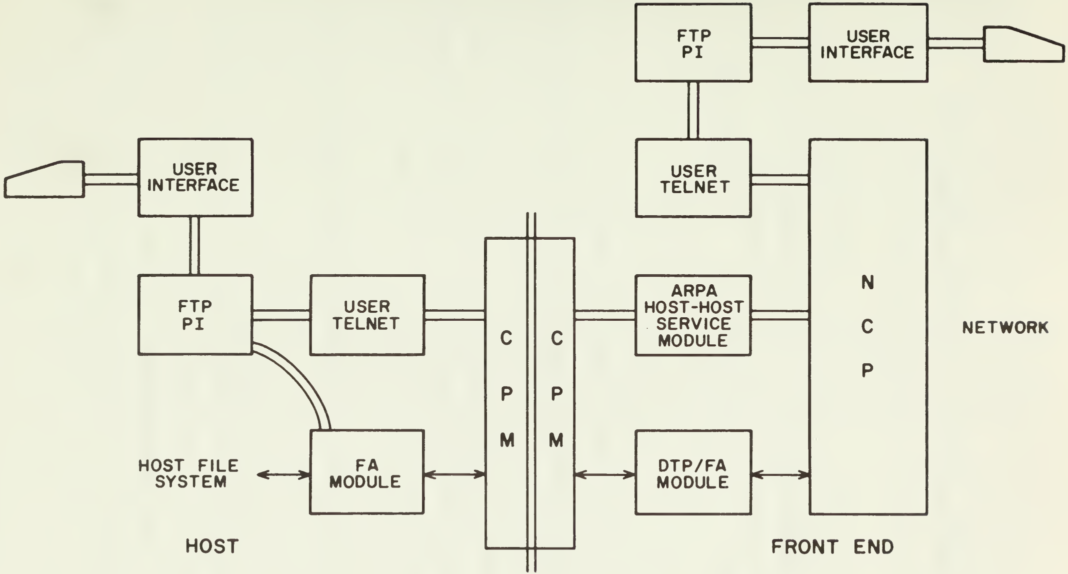


Figure 8. Data Transfer in the Front End

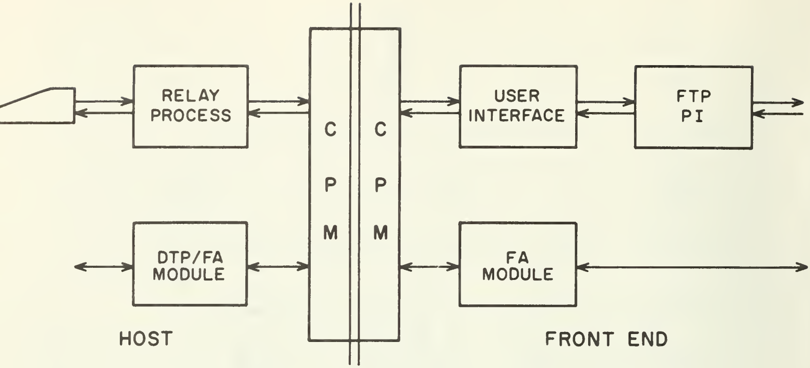


Figure 9

Data Transfer in the Host

host before the transfer can commence. Of course, the DTP must be able to report restart markers back to the front end.

6. User interface and DTP in the host. This scheme exhibits most of the advantages of its counterpart. The DTP can be initiated by the host, thus solving several problems. Restart management can be handled by either side. The front-end side of the FAS becomes very simple, since mode, type, etc., parameters could be supplied by the user interface.

7. Telnet in the front end. The only problem this variation presents is coordinating the Telnet connection and the data connection if the default sockets are used.

8. Nothing is offloaded. Everything is done through the ARPA Host-Host PSP [13]. If a file system exists in the front end, it will be necessary to duplicate the DTP in the front end. Thus, in systems where file systems exist in both, it is probably not worthwhile to consider models with the DTP in the host.

The question may occur to some, why consider putting the DTP in the host at all? The best example would be a situation where the front-end's instruction set, although very well suited for message switching, is not well-suited to the kind of bit-manipulations done by the DTP. In such a case, considerable savings may be gained by putting the DTP in the host.

Offloading the Server FTP

Unlike the situation for User FTP, the offloading of Server FTP does not follow the clear functional lines of the FTP model. We will use the model shown in figure 10 as a basic framework for the descriptions of offloading Server FTP. This model can be used to represent virtually all of the FTP offloading schemes. For the present, we have placed the DTP functions of translating and reformatting the data in the front end. Although these functions could be assigned to the host (see User FTP scheme 5), the most likely assignment is to the front end. We will consider this case later. The schemes described in this section also use a File Access-PSP to move data into the host. The requirements for this FA-PSP are virtually identical to the one described above.

Offloading Server FTP is complicated somewhat by the fact that a file system may exist on both the host and the front end. For the Server FTP this means that some well publicized convention for distinguishing these two file systems may be required in the pathname.

Interestingly enough, when allocating functions between the host FTP-PI's and the FAS module, the division appears to be that file management functions are performed by the FTP-PI and file transfer functions are performed by the FAS module.

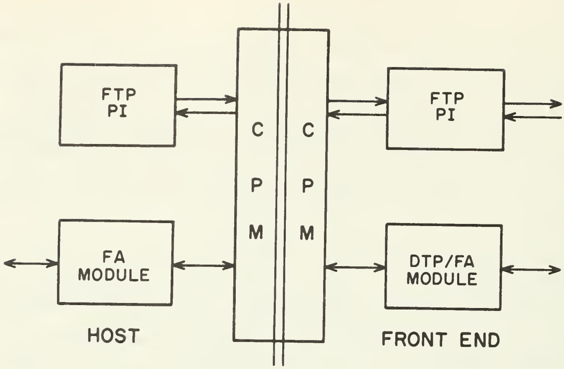


Figure 10

Server FTP Offloading

Thus, for FTP sessions that perform operations on the host file system, the following functions must be in the host FTP-PI (see table 4):

RENAME

DELETE

SITE

STATUS <arg>

LIST

NAME-LIST

ALLOCATE

Depending on the particular host operating system, it may not be possible to assign the last three commands of this list to the host FTP-PI. The LIST and NAME-LIST commands use the data connection to transfer their results to the user. Some operating systems may require that these functions be done by the host FTP-PI; but some implementations may require that access to the data connection be gained only through the host FA-PSP. Similarly, hosts which require an allocate before

APPEND	
RETRIEVE	STORE
ALLOCATE	SITE
RENAME TO	STATUS arg
RENAME FROM	LIST
DELETE	NAME-LIST
Replies to all of the above	

FTP Commands that Must Be Done in the Host

USER	ACCOUNT
PASSWORD	REINITIALIZE
BYE	BYTE
SOCKET	PASSIVE
TYPE	STRUCTURE
MODE	RESTART
STATUS	HELP
NOOP	ABORT
Replies for all of these Commands	

FTP Commands that Can Be Offloaded

Table 4. Offloading Server FTP

accepting a file may be better able to handle it if it is done through the FAS. The FTP commands for user authentication (USER, PASS, and ACCT) may or may not be offloaded depending on the security policy arrangement between the host and the front end.

Although the FTP-PI in the host must report the success or failure to the FTP-PI in the front end, it might be reasonable for the host to generate a highly encoded reply and the front end to generate a "proper" FTP reply. For configurations that support file systems on both the host and front end, this strategy has the added advantage of presenting a common reply repertoire to the network. The feasibility of this mechanism will depend heavily on the particular host and front end.

Given that these functions must be done in the host, let us consider the ones that may be in the front end:

Access Control Commands,

Transfer Parameter Commands,

RETRIEVE, STORE, APPEND, RESTART,

ABORT, HELP, STATUS, and

replies for all of the above.

It is not clear just how much of the Access Control Commands (i.e., USER, PASS, and ACCT) can be done in the FE. This will depend on the log-in arrangement between the front end and the host. It is possible that a successful log in to the front end is equivalent to a successful log in to the host. Or the front end may have to log the user into the host by initiating a host FTP-PI in the user's name before indicating success or failure of the attempt.

All of the Transfer Parameter Commands can be offloaded. These commands affect the format and encoding of the data for transmission and the manipulation of the data connection. It should be noted

that if these commands are not handled in the same machine as the DTP, then some means must be provided for communicating their values to the DTP or FAS. The DTP should be tailored to individual systems to translate the data from the transmission form (determined by the MODE, TYPE, STRUCTURE, and BYTE commands) to a form acceptable to the host. (Such translations must be invertible.)

Because data transfers are accomplished by using the FA-PSP, it is possible to do some of the processing of the RETRIEVE, STORE, APPEND, and RESTART commands in the front end. The front-end FTP-PI can interpret the command, initiate the file access process, and re-format its responses into standard FTP replies.

It is also possible for the HELP command to be done in the front end. Since the only function of this command is to return text explaining the local effects of FTP commands and implementation peculiarities, there is no reason this text cannot be stored in the front end.

There are two forms of the STATUS command. One has a pathname argument and returns information on a particular file. For files on the host, this command cannot be offloaded. The other form of the command does not have an argument and returns the status of a transfer that is in progress, or, if no transfer is in progress, the current value of all transfer parameters and the status of all connections. This function can be offloaded to the front end.

The last major aspect of offloading FTP is handling restart markers. There are two main cases to consider: data going out to the net and data coming in from the net. Let us first consider data moving out. If the front end can exert enough addressing control through the FA-PSP, it is possible to allow the FAS in the front end to insert the

restart markers, thus offloading this function from the host. The only requirement is that the FAS in the front end be able to address the same point in the file when given the restart marker and commence the transfer at that point. If this is not possible then these functions must be in the host.

For data coming into the host, it is necessary for the Server FTP to be able to report to the User FTP when the data associated with a particular marker has been entered into the file, and to associate with the user-specified markers a locally generated marker that can be used to restart the transfer at this point.

A restart function can be provided in two ways: either the FAS in the host can report the markers to the host FTP-PI, or they can be sent back to the FAS in the front end and reported to the front-end FTP-PI. If the latter approach is used, the FAS in the front end could generate the local markers and send these to the remote host, while the front-end FTP-PI handles the marker bookkeeping. If it is not possible for the FAS in the front end to assign markers, then it must be done by the host.

The File Access Process-to-Service Protocol

In each of the models just described we have postulated a utility for moving files between the host and the front end. In this section, we will review the operations that such a utility must be able to perform and discuss some of the problems involved in its design. The FA-PSP resembles the Bulk Transfer Facility described by [32] for EIN or the File Access Protocol [15]. In the models described above, we have indicated that the Data Transfer Process of FTP should be associated with one side of the FA-PSP. Typically, the front-end implementation would perform the DTP functions, including the transformation

of data from the network format to a form acceptable to the host. The host side typically would ensure that the data was written to or read from the proper file.

The front end must be able to request that the host perform the following operations:

open a file,

read data from the file,

write data to a file,

start a transfer from some point in the file (for restarts),

append to a file (this is a special case of the last point),

close a file, and

abort a transfer.

Some systems may also require the FAS module to

allocate space for a file, and

provide directory information.

In addition, the FAS module in the host must be able to provide a sufficiently rich set of responses to allow reasonable FTP replies to be generated by the FTP-PI. The FAS module must also be able to report to either the host or front-end FTP-PI that all data ahead of a restart marker have been written on the file. Some implementations may wish to have the front-end side generate restart markers for outgoing data. However, if marker assignment requires knowledge of the data (i.e., record boundaries, etc.) it may have to be done in the host.

A major concern in the use of the FA-PSP is how access control is to be provided. The most obvious solution is to have the FA-PSP "log-in" to the host on behalf of the user who initiated this session. This raises the question of whether or not there are separate usercodes

for the host and the front end. As long as the only file system being accessed is in the host, there is really no problem. The real difficulties arise when there are file systems in both host and front end, so that it may be desirable to have separate usercodes. For the time being, resolution of this problem will have to be left as an installation policy decision.

Unlike most PSP's, the FA-PSP maintains two major communication channels. On the one hand there must be a channel for the data that are moving between the host and the network, and on the other hand there must be a channel for control and state information to be passed to either the host or front-end FTP-PI. In the FA-PSP all data is sent via TRANSMIT Commands and all Control information is sent via EXECUTE Commands.

Offloading the Network RJE Protocol

This part of the study uses the ARPANET RJE Protocol (NETRJE) as a basis for discussion. Although other RJE protocols have been devised (the UCLA-CCN NETRJS [1] and the one proposed by Day and Grossman [14]), the major points that must be considered are well illustrated by the ARPANET protocol.

The ARPANET NETRJE protocol [2] is built on top of the Telnet and File Transfer Protocols. By means of the User NETRJE program on his local host, a user of NETRJE establishes a Telnet connection to a Server NETRJE process at the host where he wishes to submit a job. (See figure 11.) The user then sends to the Server process appropriate commands to log in, to describe where the job is that is to be submitted, to describe how the job is to be transferred, and to describe what is to be done with the output when the job is completed. The Server NETRJE process on the foreign host then uses its User FTP program to transfer the input file to the foreign host, and then may use FTP again later to send the output back to the user. This protocol is very general and allows a user to submit his input from one host, process it at a second, and send the output to a third.

Offloading the User RJE

For User NETRJE, the only offloading scheme (see figure 12) that is at all interesting uses the PA-PSP to get to the NETRJE user interface. (A scheme could be constructed that put the user interface in the host. Such a scheme would then use Telnet either through the PA-PSP, which would require some command-language emulation, or through the Network Virtual Terminal PSP. In either case, the resulting models are simplified versions of those considered in the User FTP section above.)

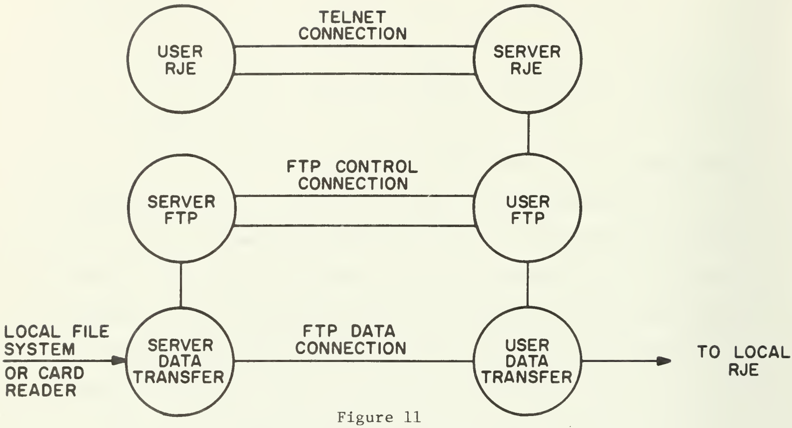


Figure 11

The Model of the ARPANET RJE Protocol

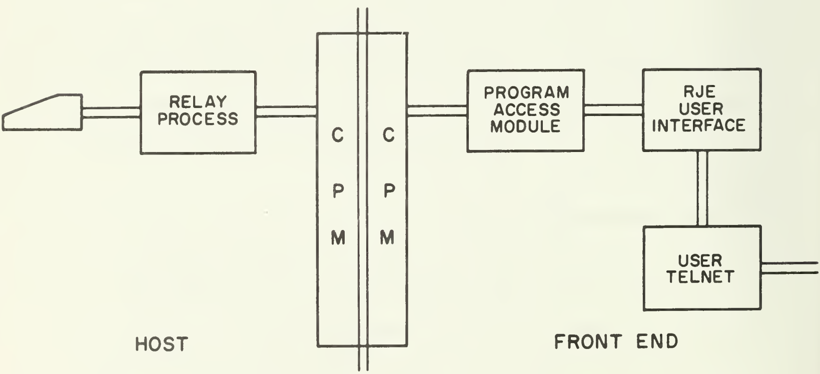


Figure 12

User RJE Offloading

Most of the offloading that can be accomplished for NETRJE consists of the offloading of FTP. However, for an offloaded FTP that uses the FA-PSP, there is an additional facility that the FA-PSP must have in order to be used with NETRJE. Since the remote host will use FTP to retrieve input and store output, the FA-PSP should be able to access the card readers (or their agents) for input requests and to access line printers or spooling queues for disposing of output.

Offloading the Server RJE

There is some difficulty in determining exactly how much of the server function can be offloaded in any general way. Many RJE or batch systems for contemporary operating systems are rather monolithic in their structure. Therefore it is not clear how separable some functions are from the whole. In fact, for some implementations Server NETRJE may be a translator between the local RJE system and the network conventions.

Assuming that functions are fairly separable, we notice that NETRJE functions can be divided into two types: those used to control the FTP and those used to control the RJE system. The FTP functions can be offloaded to as great a degree as FTP can be. Since most RJE systems are monolithic and the tables containing the information specifying the disposition of the jobs are in the host, the RJE functions in general cannot be offloaded. (See table 5.)

The offloaded Server NETRJE must be able to associate job-ID's with host pathnames or be able to direct output from the RJE system to the FAS. Also, it must be able to direct input from the FAS to the RJE system and associate it with the NETRJE session.

REINIT	OUT+
CHANGE+	ABORT (output)+
BACK+, SKIP+	STATUS
CANCEL, ALTER	

NETRJE Commands that Must be Handled in the Host

REINIT	USER, PASS*
INID, INPASS	OUTID, OUTPASS
INPUT, INPATH	ABORT
RESTART, RECOVER, HOLD	

NETRJE Commands that May be Handled in the Front End

Table 5. Server NETRJE OFFLOADING

* It may be possible to offload these commands, depending on how user authentication is handled.

\+ Some systems may spool network output to the front end, in which case these commands could be offloaded.

Offloading Teleconferencing

Most teleconferencing systems exist as programs, not protocols. The only example of a teleconferencing system that is constructed as a protocol is Calvin's [3]. In this system, the user interface for the system is located on one host and the conferencing system itself is located on another.

Even though such systems are not protocols, they are important network services. Let us consider how they may be offloaded. If the front end has its own file system, then it is possible to offload the entire teleconferencer. If no file system exists or there is insufficient file space in the front end, then all of the conferencing functions could reside in the front end and the FA-PSP could be used to write the transcript in a file on the host. Given the structure of most teleconferencing systems, it appears that most aspects of the conference management functions are sufficiently tightly related that they should not be separated. However, sophisticated teleconferencers that provide the use of such facilities as graphics displays, simulation programs and data management systems may allow greater flexibility in offloading.

Offloading Network Graphics Protocol

Although the Graphics protocol is, strictly speaking, outside the scope of this document, we will consider briefly how it may be offloaded. The ARPANET Network Graphics Protocol (NGP) is much too complex to describe in detail. It is a general purpose protocol limited to "calligraphic pictures, to moderate interaction demands, and to 'best effort' attempts to generate the required graphics" [33]. Several aspects of graphic systems, such as shading and animation, are not included because of a lack of understanding of how best to model these fairly new facilities.

To describe the NGP model we quote directly from the protocol document [33]. (The figures have been renumbered to keep the numbering consistent throughout the present document.)

> "The philosophy behind the network graphics protocol can be demonstrated by giving a model of a general-purpose graphics system and combining it with the network model of Figure 13. The model of the system is shown in Figure 14... To avoid misunderstandings, and for the sake of defining terminology, it is worthwhile to describe briefly each of the elements of Figure 14:
>
> 1. The input devices -- keyboard, stylus etc. -- are used by the operator of the application program to provide data and to control the program during execution.
>
> 2. The application data structure contains data, basically nongraphical, relating to the application program.
>
> 3. The input routines receive data from the input devices, make appropriate changes to the application data structure, and hand control to other routines.
>
> 4. The non-I/O routines analyze and modify the application data structure.
>
> 5. The output routines build a structured picture definition from data drawn from the application data structure. Effectively they define how this data may be visualized for display purposes.

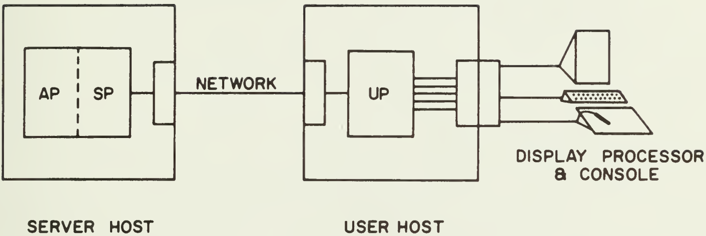
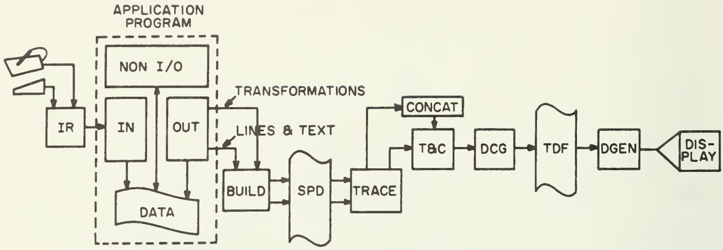


Figure 13

A graphics application program (AP) running in one computer (the Server Host) drives a display console attached to another host (the User Host).



Application Program:

IN	Input Routines
OUT	Output Routines
NON 1/0	Application Routines
DATA	Application Data Structure

Graphics Package and Hardware:

IR	Interrupt Routines
BUILD	Routines to build the SPD
SPD	Structured Picture Definition
TRACE	Routines to trace the SPD
CONCAT	Concatenation routine
T&C	Transformation and clipping routines
DCG	Display code generator
TDF	Transformed Display File
DGEN	Display generator

Figure 14

A Conceptual Model of a Graphics Application Program, a Graphics Package and Display Console

6. The structured picture definition defines the entire picture to be displayed, part or all of which may be visible on the screen. The picture definition is generally made up of a number of elements, representing parts of the picture known collectively as subpictures; subpictures may themselves be made up of other subpictures. A familiar type of subpicture is the symbol which is often used many times within a single picture or subpicture. Each reference or call to a subpicture element may denote a transformation -- scale, rotation, etc. -- to be applied to the subpicture.

7. The transformation routines apply the transformations specified in the structured picture definition and clip out information that lies off-screen. Often an arbitrary rectangular viewport is used as the clipping boundary instead of the screen edge. These routines also handle the concatenation of transformations necessitated by multi-level structured picture definitions.

8. The transformed display file is essentially what remains of the picture after transformation and clipping. It is defined in the screen's coordinate system and is generally stored in a format that allows direct refresh or regeneration of the picture. The display file contains "primitives" that specify lines, dots and text to be displayed.

9. The display generator generally includes a vector generator and a character generator, which transform the contents of the transformed display file into signals that the display's deflection system can understand.

10. The display itself.

"We shall now analyze Figure 14 to see how the system can be implemented. The diagram illustrates three information structures (an application data structure, the structured picture definition, and the transformed display file), and three processes (the output routines, the transformation, and the display generator). The design of a general-purpose graphics system requires specifying the roles of all of these elements, with the exception of the application data structure. Our examination amounts to asking: what can the hardware do, and what does the graphics programmer think he can do?...

"Most display hardware is "processor" hardware, capable of implementing some or all of the three processes in Figure 14. If, for example, a display processor is available that implements transformations and display generation, then the transformed display file is absent, and we can view the

hardware as "an interpreter of structured picture definitions." If the available hardware has fewer capabilities (e.g. no transformation ability), the picture is refreshed from the transformed display file, and the hardware is "an interpreter of transformed picture definitions." Or, if the hardware is a storage-tube terminal that does not require a display file for refreshing purposes, part of the display generation process is a software process. (However, a display file is still required, as we shall see below.)

"Determination of what the programmer can do is somewhat more subtle, because the graphics system designer has complete control of the facilities he presents to the programmer. For example, the programmer of the system shown in Figure 14 would think he was creating a structured picture definition: the output routines allow him to create, modify, and delete elements of that structure. The remaining processes (transformation and display generation) and structures (transformed display file, if it exists) are inaccessible to the programmer; in some sense, he thinks that a "display processor" is interpreting the structured picture definition in order to generate a display. He cannot determine whether a transformed display file exists, nor whether transformations are done in hardware or software, etc.

"If the structured picture definition is deleted, the programmer sees a quite different set of facilities. The output routines create (after transformation) a transformed display file. The programmer now thinks that the hardware is a "transformed display file interpreter." Again, the programmer is unaware of the details of the display generator process (for example, if the display is a storage tube terminal, the display generator is a combination of a software display-file interpreter and some hardware vector and character generators. If, on the other hand, the transformed display file is used to refresh a display, the display generator is presumably entirely in hardware).

"The discussion suggests that relative device independence can be provided if we permit two different kinds of output information: (1) information for building structured picture definitions, or (2) information for building transformed picture definitions directly. The first of these is matched to high-performance displays such as the LDS-2 or LDS-1; the second to standard refresh displays such as the IMLAC or GT-40.

"How does this relate to network protocol? We can essentially view the UP [user program] as a "display processor," i.e. an "interpreter of structured picture definitions," or an "interpreter of transformed picture definitions." The network protocol is used to build and modify a picture definition contained in the user host. Various UP options are shown in Figure 15 (the dotted lines surround those

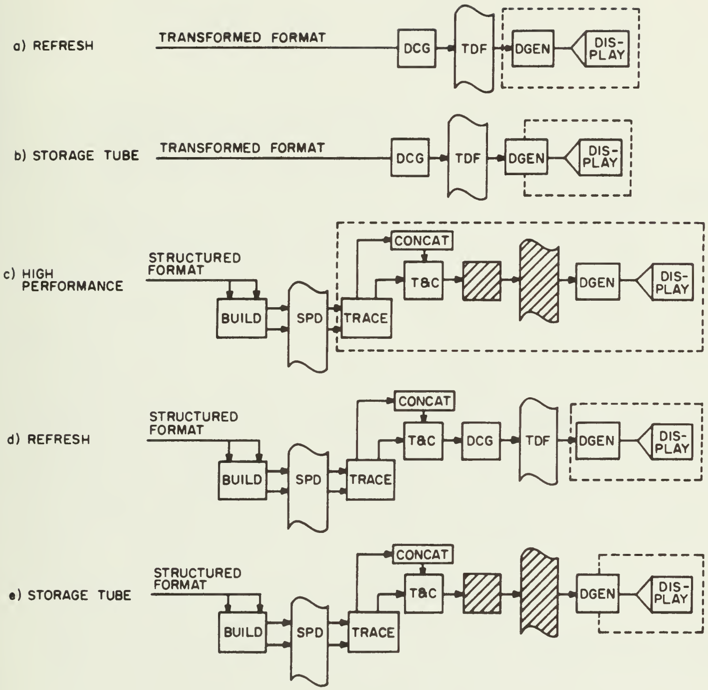


Figure 15

Various configurations for the user program.
Dashed lines surround processes implemented with hardware.
Hatched processes are omitted in the particular configuration.

processes that are implemented with display processor hardware). Figures 15a and 15b show the network used to transmit transformed information; the UP uses this information to build a display file for refreshing a display or for updating a storage-tube. Figures 15c, 15d and 15e show the network being used to transmit information for building a structured picture definition; the UP is an interpreter of a structured display file.

"Figure 16 illustrates some possibilities for the server; the server can generate either structured or transformed display information. In Figures 16a and 16b the programmer "sees" a structured picture definition; in Figure 16c a transformed picture definition.

"In the protocol, the UP tells the SP [server program] which kinds of format it can implement. It is perfectly possible for the UP to implement both -- the display images due to each of the formats are merged onto the screen.

"The two different kinds of output format can be summarized as follows:

Transformed

> The protocol is used to build and modify a set of "segments" (sometimes called "records") of a transformed display file, stored in the user host. A segment is a list of graphical primitives that specify lines, dots and text to be displayed at specific positions on the display screen. Individual segments may be deleted or replaced; they may be added to or removed from a list of segments to actually display on the screen (this is called "posting" and "unposting" segments). If a picture is composed of many segments, changes to the picture can often be made by replacing one or two segments; thus segmenting the display file helps to reduce the amount of information that must be transmitted through the network to effect a change. Considerable experience with this type of picture definition has demonstrated that device independence can easily be achieved...
>
> One advantage of transformed format is that the UP can be kept very simple; no transformations need be performed in the UH [user host]. A UP along the lines of Figure 15a should be able to be implemented in an IMLAC. The burden of transformation is left to the SH [server host], presumably a large computer quite capable of being programmed to do transformations.

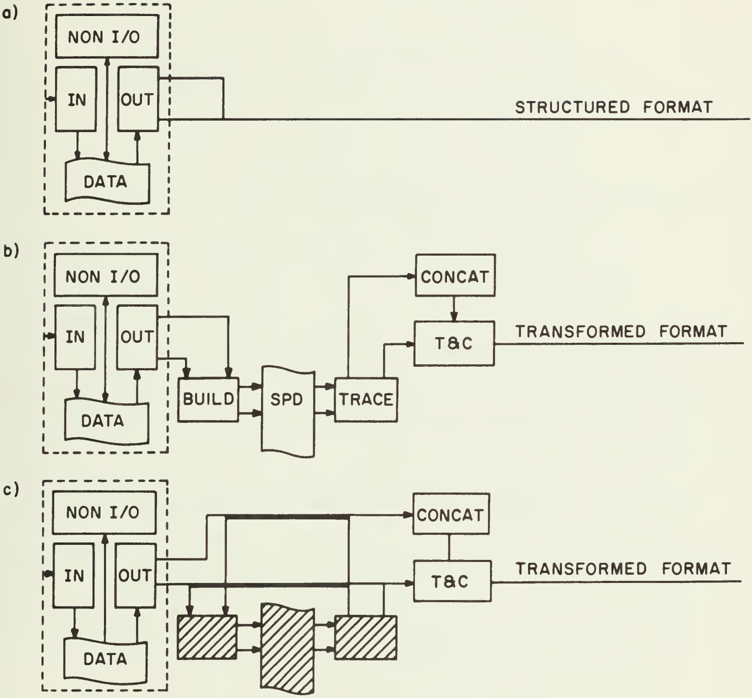


Figure 16

Various configurations for the server program.
These mate with the user program configurations of figure 15.

Structured

> The protocol is used to build and modify "figures;" each figure is a collection of "units." A unit may be a list of graphical primitives, such as lines, dots and text; or it may specify a "call" on another figure, together with a transformation to apply to the called figure. The protocol can replace individual units; altering a figure that is called in several places may cause widespread changes to the visible display. The structured format requires (in principle) even less network bandwidth for updates than does the transformed format; many updates may involve changing a single transformation (e.g. to change the viewing transformation for a three-dimensional object).
>
> Although a structured picture definition can be interpreted directly by a few display processors that have transformation ability, most UP's that implement structured format will perform the transformation in software... This implementation is relatively difficult, and certainly requires a fairly powerful UH computer."

As one can readily see, of all the protocols considered here this protocol is the most suitable for offloading. Figures 14, 15 and 16 provide us with a fairly good idea of how the protocol functions should be placed. An exact assignment of duties, however, would depend heavily on the capabilities of the graphics terminal, the host, and the front end. Some work along these lines has already been done by Van Dam [35] and Foley [18]. These two investigators have considered the problem of how to best share the load for a graphics application between a host and a dedicated satellite graphics processor. Further work would be necessary to evaluate a system consisting of a host, a network front end, and a graphics display with or without its own satellite processor.

References

1. Braden, Bob
   1971, "Interim NETRJS Specification," RFC 189, NIC #7133.

2. Bressler, R.; Guida, R.; and McKenzie, A.
   1972, "Remote Job Entry Protocol," RFC 407.

3. Calvin, James O.
   1974, "The Design and Implementation of an Interactive Teleconferencing Environment," Undergraduate thesis, Case Western Reserve.

4. Crispin, Mark
   1977, "Telnet Logout Option," RFC 727, NIC #40025.

5. Crocker, D.H. and Postel, J.B.
   1973, "Remote Controlled Transmission and Echo Telnet Option," NIC #19859.

6. Crocker, D.H.
   1974, "Telnet Output Carriage-Return Disposition Option," RFC 652, NIC #31155.

7. Crocker, D.H.
   1974, "Telnet Output Horizontal Tabstops Option," RFC 653, NIC #31156.

8. Crocker, D.H.
   1974, "Telnet Output Horizontal Tab Disposition Option," RFC 654, NIC #31157.

9. Crocker, D.H.
   1974, "Telnet Output Formfeed Disposition Option," RFC 655, NIC #31158.

10. Crocker, D.H.
    1974, "Telnet Output Vertical Tabstops Option," RFC 656, NIC #31159.

11. Crocker, D.H.
    1974, "Telnet Output Vertical Tab Disposition Option," RFC 657, NIC #31160.

12. Crocker, D.H.
    1974, "Telnet Output Linefeed Disposition Option," RFC 658, NIC #31161.

13. Crocker, D.H.
    1977, "Telnet Byte Macro Option," RFC 729, NIC #40036.

14. Day, J.D. and Grossman, G.R.
1977, "An RJE Protocol for a Resource Sharing Network,
RFC 715.

15. Day, J.D.
1973, "A Proposal for a File Access Protocol," RFC 520.

16. Day, J.D.
1976, "Cost and Response Time Models for Network Applications," Internal Modeling Memo #14, Center for Advanced Computation, University of Illinois at Urbana-Champaign.

17. Day, J.D.
1977, "Telnet Data Entry Terminal Option," RFC 731, NIC #40652.

18. Foley, James and Brownlee, Edward
1974, "A Model of Distributed Processing in Computer Networks with Application to Satellite Graphics," Proceedings of ICCC.

19. Grossman, G.R.
1977, The Host-to-Front-End Protocol, CAC Document #219, Center for Advanced Computation, University of Illinois at Urbana-Champaign.

20. Mock, Tovas and Postel, Jon
1974, "Telnet Extended ASCII Option," RFC 698, NIC #32964.

21. Neigus, N.
1973, "The File Transfer Protocol," RFC 542.

22. Neigus, N.; Pogran, K.; and Postel, J.
1974, "A New Schema for FTP Reply Codes," RFC 640.

23. Postel, Jon
1973, "Telnet Protocol Specification," NIC #15372.

24. Postel, Jon
1973, "Telnet Option Specifications," NIC #15373.

25. Postel, Jon
1973, "Telnet Binary Transmission Option," NIC #15389.

26. Postel, Jon
1973, "Telnet Echo Option," NIC #15390.

27. Postel, Jon
1973, "Telnet Reconnection Option," NIC #15391.

28. Postel, Jon
1973, "Telnet Suppress Go Ahead Option," NIC #15392.

29. Postel, Jon
1973, "Telnet Status Option," NIC #16237.

30. Postel, Jon
1973, "Telnet Timing Mark Option," NIC #16238.

31. Postel, Jon
1973, "Telnet Extended-Option-List Option," NIC 16239.

32. Schicker, Peter; Duenki, Ann; and Baechi, W.
1975, "Bulk Transfer Function," INWG Protocol #31, EIN/ZHR/75/20.

33. Sproull, Robert F. and Thomas, Elaine L.
1974, "A Network Graphics Protocol," NIC #24308.

34. Stone, Harold
1977, "Multiprocessor Scheduling with the Aid of Network Flow Diagrams," IEEE Transactions on Software Engineering, Volume SE-3, No. 1.

35. Van Dam, Andries; Stabler, George M.; and Harrington, Richard J.
1974, "Intelligent Satellites for Satellite Graphics," Proceedings of IEEE, Volume 62, No. 4.

36. Walden, David C.
1973, "Telnet Output Line Width Option," NIC #20196.

37. Walden, David C.
1973, "Telnet Output Page Size Option," NIC #20197.

Appendix 1

Program Access Process-to-Service Protocol Specification

CAC Technical Memorandum #81

by

John Day
Steve Holmgren

Program Access

Process-to-Service Protocol

Specification

I. Service Code Number

2

II. Description of the Service

This document is a process-to-service protocol specification as defined in the Host-to-Front-End Protocol (HFP) Specification [ARPANET RFC 710]. It describes a process-to-service protocol for the execution of, or attachment to, arbitrary programs in the front end. The intent is to provide a general mechanism that will allow the host to access terminal-oriented front-end services. Examples of such services are User Telnet and teleconferencing.

This protocol allows terminals directly connected to the host to appear to the front end as if they were local front-end terminals. This strategy for handling host terminals has two advantages. Users are confronted with a common command language and services can be almost completely offloaded to the front end.

The protocol described here assumes that the program access service itself is completely offloaded to the front end. The only software remaining in the host is a relay process that passes properly formatted data between a host terminal or process and the program-access-service module in the front end. HFP

Messages are used for this data transmission.

For some implementations, the relay process software must

1. ensure that security information is properly transmitted in BEGIN Commands (this may require querying the user for usercode and password);

2. properly handle half-duplex terminals (via the GO AHEAD facility);

3. transform interrupt characters into EXECUTE Commands;

4. flush data when instructed to do so;

5. accept data from terminals with differing character sets and properly transform them for transmission to the front end;

6. divert binary or character data to auxiliary devices; and

7. modify terminal parameters that control echoing.

The more items from this list that a PA-PSP host implementation must support, the less the savings to be gained by using this PSP. Since most implementations will give the user of this protocol access to the front end at the command language level, it will be inappropiate for use by processes.

The only support that the relay process requires is the ability to access the front end and to be accessed by host terminals or processes. Some systems may experience difficulty providing one interface that can handle both.

The program-access-service module must have the ability to attach to and to execute user-level programs within the front

end and pass data to and from the user-level programs and the channel protocol module (CPM).

The program access process-to-service protocol uses HFP Messages to carry information between the relay process and the service module. A brief summary of the function of specific Message types follows.

BEGIN Command
from the process
requests access to a program in the front end.

BEGIN Response
from the service module
a) confirms that access has been accomplished or
b) reports the reason for failure.

END Command
from either side
requests that communication with the other side be terminated.

END Response
from either side
acknowledges that an END Command has been received and that the proper action has been taken.

TRANSMIT Command
from either side
transfers data to the other side.

TRANSMIT Response
from either side
acknowledges receipt of data from the other side.

EXECUTE Command and Response
are not used in this protocol.

The details of the use of HFP Messages follow.

III. Message Use

III A. BEGIN Command

III A 1. When sent

When a terminal or process in the host wishes to execute a front-end program, which may be a service in its own right, it will start up or attach to a host relay process. The relay process will then send a BEGIN Command to the program-access-service module in the front end. The TEXT of the BEGIN Command will contain a front-end command line formatted to installation standards. The command line will specify the front-end program to be executed and will provide any required parameters.

III A 2. Action on receipt

When a BEGIN Command is received by the program-access-service module in the front end, the module will attempt to execute or attach to the specified program. Once the service module has completed a connection to the program, it should issue a BEGIN Response to the host relay process. Subsequent communications on this channel from the relay process to the service will be directed to the program.

III A 3. TEXT field syntax

```
Security:  VARIABLE(?)
Command:  VARIABLE(?)
```

III A 4. TEXT field semantics

III A 4 a. Security

This field can be used by the front end to verify the validity of the user and his access to the program. The actual contents of this field are installation dependent. If the security field is empty, the effect is to connect the user to the front end at the pre-login stage. The Command field will be ignored in this case. Note: At this writing, not enough is known about security in networks and front ends to make any statement about the utility of validation at this point in the system. This field is included in case some systems require validation at this point.

III A 4 b. Command

For most front ends the Program Access PSP provides general access to the front end command language or a set of procedures callable by host users. The command field will usually contain a string of

characters to invoke that command or procedure along with any parameters in a form similar to the way it would be typed by a user.

Example:
run telnet

If the Command field is empty but the Security field is not, then the effect is to log the user in to the front end.

III B. BEGIN Response

III B 1. When sent

The BEGIN Response is generated by the service when a determination has been made as to the success or failure of the BEGIN Command. The Status field in the HEADER will indicate the success or failure.

III B 2. Action on receipt

When the process receives a BEGIN Response indicating a successful connection, it may proceed to send and receive data on the connection. When the process receives a BEGIN Response indicating an unsuccessful connection, it should perform the appropriate error recovery.

III B 3. TEXT field syntax

Result: FIXED(3)

III B 4. TEXT field semantics

III B 4 a. Result

If the BEGIN Command was successful, this field will have a value of zero. A non-zero value indicates failure. The particular value indicates the reason for failure. The Result values and their meanings are:

0 Success
1 Access to program denied
2 Illegal Command field format
3 Program not found
4 Front end terminating service

III C. END Command

III C 1. When sent

The END Command may be issued by the program-access-service module or the host relay process. If sent by the relay process, it usually indicates that the user has requested that the front-end program be terminated and the communication path be destroyed. The service may send an END Command because of events external to the operation of the relay process and the program or in response to the termination of the front-end program.

III C 2. Action on receipt

When the relay process receives an END Command, it should acknowledge it promptly with an END Response and clean up any data structures associated with the communication path. When the program-access-service module receives an END Command, it should generate an END Response and terminate the process associated with the Channel.

III C 3. TEXT field syntax

Cause: FIXED(3)

III C 4. TEXT field semantics

III C 3 a. Cause

This field indicates the reason for termination of communication. The values of the field and their meanings are:

0	Normal completion of the program
1	Program failure
2	User requested termination
3	Unknown
4	Front end terminating service

III D. END Response

III D 1. When sent

The END Response is sent either by the relay process or by the program-access-service module to acknowledge that an END Command has been received and the proper action taken.

III D 2. Action on receipt

When an END Response is received, all dialogue over the associated logical channel is complete. All Commands over the logical channel will be ignored until the receipt of another BEGIN Command referencing that channel.

III D 3. TEXT field syntax

The TEXT field in the END Response is not used in this process-to-service protocol. The contents of the field will be ignored.

III E. TRANSMIT Command

The description below assumes that a Go-Ahead facility is needed for the host relay process to handle half-duplex terminals. Not all installations will need this facility. Those that do not should set the Control field in the TEXT to zero and ignore the specification of how this field is interpreted.

III E 1. When sent

The TRANSMIT Command may be sent by either side to transfer data. Data may be sent only when the communication path is ESTABLISHED and the flow control discipline permits. (See the HFP specification.)

III E 2. Action on receipt

III E 2 a. By the relay process

When the relay process receives a TRANSMIT Command, it will relay the contents of the Data field to the host process or terminal. If the Go-Ahead bit is set to 0 for data going to the host, then the host may assume that the front end has completed sending and the host may now send any data it has. A Go-Ahead bit set to 1 indicates that the sender has more data to send and the receiver must wait.

III E 2 b. By the program-access-service module

When the service module receives a TRANSMIT Command, it will relay the contents of the Data field to the program. If the Go-Ahead bit is set to 0 for data going to the front end, then the front end may assume the host has completed sending and the front end may send any data it has. A Go-Ahead bit set to 1 indicates that the sender has more data to send and the receiver must wait.

III E 3. TEXT field syntax

```
Control: FIXED(1)
   Go-Ahead/More Data
Data: VARIABLE(?)
```

III E 4. TEXT field semantics

III E 4 a. Control

The single bit in this field indicates whether or not the sender of the TRANSMIT Command has completed sending data. It is used to facilitate handling half-duplex terminals connected to the relay process. The

receiver of a TRANSMIT Command with the Go-Ahead bit set to 0 (default) should assume the sender has completed sending data and that the receiver is now free to send any data that it has. If the bit is set to 1 the sender has more data to send and the receiver should not send data at this time.

III F. TRANSMIT Response

The TRANSMIT Response is used as described in the HFP specification.

III G. SIGNAL Command

The SIGNAL Command is not used in the process-to-service protocol. SIGNAL Commands may be generated by process or service to request that the CPM carry out the appropriate action on the logical channel. (See HFP specification.) Thus, the effect of the SIGNAL Command is restricted to the logical channel (HFP) level.

III H. SIGNAL Response

The SIGNAL Response is not used by either process or service.

III I. EXECUTE Command

The EXECUTE Command is not used in this protocol. If an EXECUTE Command is received, an EXECUTE Response with Status equals 67 (Unused Command) should be sent.

III J. EXECUTE Response

The EXECUTE Response is not used in this protocol except as noted above.

IV. Scenario

Let us consider how the Program Access PSP would be used to offload User Telnet. Let us assume that the host relay process does not have to implement any of the special functions noted in Section II. The user at a terminal connected to the host types:

"connect ILL-CAC"

The connect command starts the relay process and hands it the parameter ILL-CAC. The relay process then generates a BEGIN Command:

<BEGIN><C>telnet ILL-CAC

(This installation does not use the security field.) The BEGIN Command is passed to the front end. The Program Access Service in the front end invokes the command line present in the TEXT field and sends a BEGIN Response to the host:

<BEGIN><C><result=success>

The User Telnet program is started and it attempts to open a connection to ILL-NTS, and sends a message to the user which the Program Access Service passes on to the host:

<TRANSMIT><C>Attempting connection to ILL-NTS

At this point the User Telnet program thinks it is talking to a user local to the front end. The Program Access Service maintains this charade passing messages back and forth to the user connected to the host. From the point of view of the user he is directly connected to the front end.

Appendix 2

File Access
Process-to-Service
Protocol Specification

CAC Technical Memorandum #102

by

John Day

File Access

Process-to-Service Protocol

Specification

I. Service Code Number

6

II. Description of the Service

This document is a process-to-service protocol specification as defined in the Host-to-Front-End Protocol (HFP) Specification [ARPANET RFC 710]. This protocol provides a general utility for file access between the host and the front end. This protocol is also used for offloading the data transfer portion of a network file transfer protocol (either user or server). This protocol also provides a general utility for file access between the host and the front end.

The file access process-to-service protocol differs from most others in that the distinction between the host side and the front-end side disappears. For this process-to-service protocol, one distinguishes between the side requesting service (the using side) and the side providing service (the serving side). No distinction is made as to which one is in the host and which in the front end. In fact, a file access module (FAM) may be constructed in such a way as to be both using and serving, depending on how each channel is established, (i.e., on who sends each BEGIN Command).

This specification describes a general file access utility that allows a process in one system to open a file and read or write data to or from any point in that file. The protocol provides separate indices into the file for reading and writing. These are called the read pointer and the write pointer. Although the ability to address within a file makes restart markers superfluous, protocols that don't explicitly allow addressing may require their use. Since restart marker "bookkeeping" might be offloaded, a set of operations are provided for controlling it. Not all fields listed in the descriptions of message use will be used in every case. Exactly what fields are present in the various Commands will depend on the offloading application. Details of FTP offloading schemes using the file access process-to-service protocol are described in detail in CAC Document 230 [1].

The facilities required by this process-to-service protocol depend heavily on the particular use being made of it. The maximum requirements would be access to the file systems of both the host and the front end, and implementations of FTP, Telnet, and an NCP in the front end (in the ARPANET case). The minimum requirements for the ARPANET case would be an NCP in the front end.

The file access process-to-service protocol uses HFP Messages to carry information between a file access module in the host and a file access module in the front end. A brief summary of the function of specific Message types follows:

<u>BEGIN Command</u>
from the using file access module
requests that service be provided.

<u>BEGIN Response</u>
from the serving file access module
a) confirms the establishment of service, or
b) reports the reason for failure.

<u>END Command</u>
from either file access module
requests the termination of service and indicates the conditions for termination.

<u>END Response</u>
from either file access module
acknowledges the termination of service.

<u>TRANSMIT Command</u>
from either file access module
carries data between the using and serving file access modules.

<u>TRANSMIT Response</u>
from either file access module
acknowledges the receipt of data.

<u>EXECUTE Command</u>
from the using file access module
requests of the serving file access module that
a) data transfer parameters be set;
b) a file be opened;
c) data be read from a file;
d) data be written to a file;
e) a read pointer position be set;
f) a write pointer position be set;
g) a network connection be opened;

from either file access module (depending on which is data source and which is data receiver)
a) sets a restart marker window width,
b) acknowledges a restart marker, or
c) aborts a transfer.

<u>EXECUTE Response</u>
from either file access module
a) acknowledges the completion of actions requested via an EXECUTE Command.
b) reports the reason for failure of an EXECUTE Command; or
c) reports on the status of the network connection.

III. Message Use

III A. BEGIN Command

III A 1. When sent

A BEGIN Command is sent by the using file access module (UFAM) to the serving file access module (SFAM) to request that a channel be established for the transfer of one or more files between the host and the front end.

III A 2. Action on receipt

When a SFAM receives a BEGIN Command, it will evaluate its ability to satisfy the request specified by the parameters in the TEXT field and reply accordingly.

III A 3. TEXT field syntax

```
Security:                   VARIABLE(?)
File Parameters:            COMPLEX(?)
   Pathname:                   VARIABLE(?)
   Direction:                  FIXED(2)
   Position:                   FIXED( )
   Size:                       FIXED( )
   Record size:                FIXED( )
Data Parameters:            COMPLEX(4)
   Type:                       FIXED(3)
   Mode:                       FIXED(2)
   Structure:                  FIXED(1)
   Byte size:                  FIXED(8)
Connection Parameters:      COMPLEX(8)
   Control-bits:                  FIXED(5)
      Init/Listen
      Socket/Channel
      Duplex/Simplex
      Relative/Absolute
      Unattachable/Attachable
   Host:                          FIXED(32)
   Foreign Socket:                FIXED(32)
   Messages:                      FIXED(16)
   Bits:                          FIXED(32)
   Local Socket:                  FIXED(32)
   Timeout:                       FIXED(16)
   Byte Size:                     FIXED(8)
```

III A 4. TEXT field semantics

Since the file access process-to-service protocol may be used in a variety of offloading schemes, the parameters of the BEGIN Command are quite extensive. They fall into four categories: Security, File Parameters, Data Transfer Parameters, and Connection Parameters. These groups of

parameters will be present if the UFAM is remotely controlling the functions of validating the user, manipulating files, controlling the FTP Data Transfer Process, or establishing network connections, respectively.

III A 4 a. Security

This field can be used to validate the requestor's access privileges to the file or network resources. It is particularly important to limit access to network security objects such as local socket 1. The substructure and content of this field are installation dependent.

NOTE: At this writing, not enough is known about security in networks and front ends to make any statement about the utility of validation at this point in the system. This field is included in case some systems require validation at this point.

III A 4 b. File Parameters

These parameters specify what file is to be transferred, in what direction, where the transfer is to begin, and how long the file is. These parameters will be present if the UFAM is in the front end and the file is in the host, or vice versa.

III A 4 b (1). Pathname

This field contains a locally recognizable file name for the file which is to be transferred into or out of the system in which the SFAM resides. If this field is empty, then the pathname will be specified at a later time. Note: It is an error to issue a Read variant of the EXECUTE Command (see below) before specifying a pathname.

III A 4 b (2). Direction

This field specifies the direction of the transfer. The values are:

1 data is transferred toward the SFAM (write)
2 data is transferred toward the UFAM (read)
3 data is transferred in both directions

The default value of this field is 3 (both).

III A 4 b (3). Position

The contents of this field specify where within the file the transfer is to begin. This facility is

used for restarting transfers and randomly accessing files. The position pointer that is set is determined by the contents of the direction field. If the direction field specifies "both", then both the read and write pointers are set to the position indicated. The width of this field and the units represented by it are installation parameters.

III A 4 b (4). Size

The contents of this field are used to indicate the size of the file to be written into the file system in the SFAM's computer. Some systems require that space be allocated for a file before beginning the transfer. The width of the field and the units represented by it are installation parameters.

III A 4 b (5). Record size

The contents of this field specify the maximum record size of the file to be transferred. This field will only be present if such specification is required by the system and the file is record-structured. The width of the field and units represented by it are installation parameters.

III A 4 c. Data Transfer Parameters

For offloading schemes in which the UFAM must remotely control the FTP data transfer process, these parameters are used to specify how the data is expected to arrive from the network. It is assumed that the host and front end have agreed on what form data will be presented by each to the other.

III A 4 c (1). Type

This field indicates the type of data to be sent in accordance with the classifications of the ARPANET FTP. The values of the field and their meanings are:

0	ASCII - Nonprint
1	ASCII - Telnet
2	ASCII - Carriage Control (ASA)
4	EBCDIC - Nonprint
5	EBCDIC - Telnet
6	EBCDIC - Carriage Control (ASA)
7	Image

The default value of this field is zero (ASCII - Nonprint).

III A 4 c (2). Mode

The contents of this field indicate the mode in which the data is transferred over the network in accordance with the categories used in the ARPANET FTP. The values of the field are:

0 Stream
1 Blocked
2 Compressed

The default value of the field is zero (Stream).

III A 4 c (3). Structure

The contents of this field indicate the structure of the file to be transferred. The structures supported are those used by the ARPANET FTP. The values are:

0 File
1 Record

The default value of the field is zero (File structure).

III A 4 c (4). Byte size

This field contains the byte size for data transfer on the connection. If this field contains zero (default), the byte size will be 8 bits.

III A 4 d. Connection parameters

These parameters are used for offloading schemes in which the UFAM must remotely specify the nature of the network connection. For example, if the only parts of FTP that are offloaded are Telnet and the Data Transfer Process, the UFAM in the host would have to be able to specify the parameters for opening the data connection.

III A 4 d (1). Control-bits

The bits in this field govern the kind of connection to be established, the method of its establishment, and the interpretation of the parameters.

III A 4 d (1) (a). Init/Listen

When this bit is zero (default), the connection is to be actively initiated by the NCP. When this bit is one, the connection is to be

passively "listened" for by the NCP.

III A 4 d (1) (b). Socket/Channel

If this bit is zero, the logical channel from the UFAM is to be established to a new ARPANET connection. If this bit is one, the logical channel from the UFAM is to be attached to an already existing HFP channel (e.g., a channel already established to the host-host service module and therefore providing access to an existing network connection.) The default value is zero.

III A 4 d (1) (c). Duplex/Simplex

When this bit is zero (default), the connection is to be full duplex and is therefore to consist of two ARPANET connections (one in each direction). When this bit is one, the connection is to be unidirectional according to the gender of the socket numbers supplied in the Foreign Socket and/or Local Socket fields. The use of this bit in specifying connection sockets is detailed in table 1.

III A 4 d (1) (d). Relative/Absolute

This field affects the value of the socket for the local connection. See III A 4 d (6) for details.

III A 4 d (1) (e). Unattachable/Attachable

When this bit is one, the SFAM is requested to make the logical channel available for attachment by other service modules within the front end (such as a data transformation service). If this bit is zero (default), the SFAM is not requested to make the logical channel available for attachment within the front end.

III A 4 d (2). Host

This field specifies the network host with which the connection is to be made. A value of zero in this field indicates that the connection may be established with any host on the network. A zero value is valid only if the Init/Listen bit is one (Listen).

The Host field is parsed as;

Network:	FIXED(8)
IMP:	FIXED(16)
Host-at-IMP:	FIXED(8)

III A 4 d (3). Foreign Socket

This field is used as follows to construct the <u>effective foreign socket</u> (e.f.s.) for making the connection.

Case 1. Foreign Socket field = zero. Init/Listen bit = one. Foreign host chooses the e.f.s.

Case 2. Foreign Socket field = zero. Init/Listen bit = zero. The e.f.s. is one.

Case 3. Foreign Socket field ≠ zero. The e.f.s. is the contents of the Foreign Socket field.

The default value of this field is zero. The relationship between the e.f.s. and the foreign sockets for the connection is defined in table 1.

III A 4 d (4). Messages

This field and the Bits field allow the UFAM to indicate to the SFAM the flow rate desirable on the data connection. The SFAM may use this information in any way its implementors deem desirable.

This field contains the number of ARPANET Host-Host messages the UFAM suggests be allocated on the data connection. If the value of this field is zero (default), the SFAM chooses the number of messages on its own. In some cases use of this field must be coordinated with similar information provided through other protocols (e.g., User and Server FTP process-to-service protocols).

III A 4 d (5). Bits

This field contains the number of bits the UFAM suggests be allocated for the data connection. If the value of this field is zero (default), the SFAM chooses the number of bits on its own. In some cases use of this field must be coordinated with similar information provided through other protocols (e.g., User and Server FTP process-to-service protocols).

III A 4 d (6). Local Socket

This field is used as follows to construct the <u>effective local socket</u> (e.l.s.) for making the connection.

Case 1. Relative/Absolute bit = zero, Local Socket field ≠ zero. In this case, the contents of the Group field in the HEADER will be concatenated with the low-order 20 bits of the Local Socket field to form the e.l.s.

Case 2. Relative/Absolute = zero, Local Socket field = zero. In this case, the SFAM will choose a number whose high order 12 bits are equal to the Group field in the HEADER.

Case 3. Relative/Absolute = one. In this case the effective local socket is equal to the contents of the Local Socket field.

The relationship between the e.l.s. and the local sockets for the connection is defined in table 1.

III A 4 d (7). Timeout

This field contains the maximum length of time, in seconds, that the SFAM is to attempt to establish the connection (in the absence of an error or exceptional condition). If the SFAM has been unable to establish the connection within this time period, it is to abandon the attempt. If this field contains zero (default), the time period is to be infinite.

III A 4 d (8). Byte Size

This field contains the byte size for data transfer on the connection. If this field contains zero (default), the byte size will be 8 bits.

Table 1

Control Bit	Foreign Sockets for the connection	Local Sockets for the connection
Duplex	e.f.s. and e.f.s. +1	e.l.s. and e.l.s. + 1
Simplex	e.f.s.	e.l.s.

III B. BEGIN Response

III B 1. When sent

A BEGIN Response is sent by the SFAM to report the outcome of an attempt to access a file or establish a transfer requested by a BEGIN Command. The Status field in the HEADER will indicate whether the attempt was successful or unsuccessful.

III B 2. Action on receipt

When the UFAM receives a BEGIN Response indicating a successful connection, it may proceed to use the channel. When the UFAM receives a BEGIN Response indicating an unsuccessful connection, it should perform the appropriate error recovery.

III B 3. TEXT field syntax

Security:	VARIABLE(?)
File Results:	FIXED(4)
Data Results:	FIXED(5)
Connection Results:	COMPLEX(8)
Connection State:	FIXED(4)
Host:	FIXED(32)
Foreign Socket:	FIXED(32)
Messages:	FIXED(16)
Bits:	FIXED(32)
Local Socket:	FIXED(32)

III B 4. TEXT field semantics

III B 4 a. Security

This field may be used to validate the access privileges of the user. The default value of the field is empty. Note: At this time, very little work has been done on the organization of security for networks or these kinds of systems. It is not clear where access control should be placed. This field has been included in case some systems require validation at this point.

III B 4 b. File results

If the BEGIN Command was not successful (as indicated by the Status field in the HEADER), this field will contain an encoded identification of any problems with the File Parameters. The values are:

1	File not found
2	File name not allowed
3	File too large

4	File not presently available
5	Record size too long
6	Position information not valid
7	Direction not supported
8	Illegal parameter combination

III B 4 c. Data transfer results

If the BEGIN was not successful (as indicated by the Status field of the HEADER), this field will contain an encoded identification of any errors in the Data Transfer Parameters. The values are:

1	Improper Type
2	Improper Mode
4	Improper Structure
8	Byte size not supported
16	Illegal parameter combination

If there was more than one error, the associated codes may be added to form the value of the field. This addition amounts to just setting the proper bits. If the reason is "illegal parameter combination" (16), the SFAM should also try to indicate the parameters in conflict by setting the appropriate bits.

III B 4 d. Connection results

If no connection was associated with the BEGIN Command, these fields will not be present.

III B 4 d (1). Connection-state

If the connection attempt was successful (as indicated by the Status field in the HEADER), this field will contain the value 1 which will be compatible with the "open" state value defined for this field in the TEXT of the EXECUTE Response (See III K 4 g (1).)

If the connection attempt was unsuccessful, this field contains an encoded reason for the failure. The codes are:

0	Illegal control bit combination
1	Invalid local socket
2	Invalid host
3	Byte size not supported by front end
4	Byte size not supported by foreign host
5	Connection attempt timeout
6	Network not available
7	Foreign host dead

8	Connection refused
9	Front end terminating service
10	Improper access

III B 4 d (2). Host

If the connection attempt was successful, this field contains the ARPANET host address of the host to which the connection was established (see III A 4 d (2)).

III B 4 d (3). Foreign Socket

If the connection attempt was successful, this field contains the socket number at the foreign host to which the connection was established.

III B 4 d (4). Messages and Bits

If the connection attempt was successful, these fields contain the flow control parameters for the connection at the time the BEGIN Response was generated.

III B 4 d (5). Local Socket

If the connection attempt was successful, this field contains the local socket number for the connection.

III B 4 d (6). Byte Size

If the connection attempt was successful, this field contains the byte size, in bits, for the connection.

III C. END Command

III C 1. When sent

An END Command is sent by either file access module to terminate a session on a particular channel.

III C 2. Action upon receipt

When a file access module receives an END Command, it should finish writing any data it has in hand onto the file or network connection (unless Flush away (bit 1) is set in the Control field of the HEADER), close the file or network connection, and clean up whatever loose ends there may be. When all is in order (files and connections closed, etc.), the FAM should send an END Response.

III C 3. TEXT field syntax

Cause: FIXED(3)

III C 4. TEXT field semantics

This field contains an encoded reason for sending the END Command. The codes are:

0	Normal completion (default)
1	Network/process failure
2	Foreign host failure
3	User requested termination
4	Front end terminating service
5	Improper access

III D. END Response

III D 1. When sent

The END response is sent by either FAM to acknowledge the receipt of an END Command and to complete the termination of the associated logical channel.

III D 2. Action on receipt

When a FAM receives the END Response, it should consider the logical channel to be terminated.

III D 3. TEXT field syntax

The TEXT field in the END Response is not used in this protocol.

III E. TRANSMIT Command

III E 1. When sent

III E 1 a. By the FAM in the host

The TRANSMIT Command is sent by the FAM in the host to pass data and restart markers to the FAM in the front end for transmission over the network to the foreign host, or to the file system in the front end.

III E 1 b. By the FAM in the front end

The TRANSMIT Command is sent by the FAM in the front end to pass data from the network or the front-end's file system to the FAM in the host.

III E 2. Action on receipt

III E 2 a. By the FAM in the host

When the FAM in the host receives a TRANSMIT Command, it will treat the contents of the Data field as data to be processed as directed by the associated EXECUTE Command.

III E 2 b. By the FAM in the front end

When the FAM in the front end receives a TRANSMIT Command, it will treat the contents of the Data field as data to be transmitted over the network, or to the local file system.

III E 3. TEXT field syntax

```
Special Conditions:  FIXED(2)
   No-EOT/EOT
   No-EOF/EOF
Marker:              VARIABLE(?)
Data:                VARIABLE(?)
```

III E 4. TEXT field semantics

III E 4 a. Special Conditions

This field indicates special conditions relevant to the transmission of the data.

III E 4 a (1). No-EOT/EOT

When this bit is zero (default), there is more data to follow in subsequent TRANSMIT Commands. When this bit is one, it indicates that the data in this

TRANSMIT Command completes a read or write operation.

III E 4 a (2). No-EOF/EOF

When this bit is zero (default), no end of file condition exists. When this bit is one, it indicates that the data in this TRANSMIT Command completes the file. Note: If this bit is set, the No-EOT/EOT bit will be set also.

III E 4 b. Marker

This field contains the restart marker. Since the restart marker must be at the front of the message, the FAM will have to place data into messages in such a way that each marker falls at the beginning of a new message. The form of the restart marker is left to the decision of the installation. This field can also be used to synchronize read and write EXECUTE Commands with the data they refer to.

III E 4 c. Data

This field contains the data to be passed over the channel.

III F. Transmit Response

The TRANSMIT Response is used as described in the HFP specification.

III G. SIGNAL Command

The SIGNAL Command is not used in the process-to-service protocol. SIGNAL Commands may be generated by process or service to request that the CPM carry out the appropriate action on the logical channel. (See HFP specification.) Thus, the effect of the SIGNAL Command is restricted to the logical channel (HFP) level.

III H. SIGNAL Response

The SIGNAL Response is not used by either process or service.

III J. EXECUTE Command

III J 1. When sent

III J 1 a. By the UFAM

The UFAM sends an EXECUTE Command to request that the SFAM

(1) set Data Transfer Parameters,
(2) open a file,
(3) read from a file,
(4) write to a file
(5) move the read pointer in the file,
(6) move the write pointer,
(7) open a network connection, or
(8) return a description of the status of the connection.

III J 1 b. By UFAM or the SFAM

Either FAM may send EXECUTE Commands to the other

(1) to set the marker window width,
(2) to abort a transfer, or
(3) to acknowledge a marker

Which FAM may send these commands depends upon which one is the data source and which the data receiver. The data source sets the marker window width. Only the receiver sends marker acknowledgements. Either side may abort the transfer.

III J 2. Action upon receipt

When the SFAM or UFAM receives an EXECUTE Command it should decode it and attempt to perform the action indicated.

III J 3. TEXT field syntax

Code:	FIXED(4)
Parameters:	VARIABLE(?)

III J 4. TEXT field semantics

III J 4 a. Code

This field contains an encoding of the type of request. The codes are:

0	Set Data Transfer Parameters
1	Open

2	Read
3	Write
4	Set read position
5	Set write position
6	Abort
7	Open network connection
8	Set marker window
9	Return connection status
10	Marker acknowledgement

III J 4 b. Parameters

This field contains any parameters required to perform the request indicated in the Code field. A discussion of these parameters follows.

III J 4 b (1). Set Data Transfer Parameters

For offloading schemes where the UFAM must remotely control the FTP data transfer process, these parameters are used to specify how the data is to be formatted for transmission over the network. It is assumed that the host and front end have agreed on the form in which data will be presented to each other.

III J 4 b (1) (a). Parameter field syntax

Type:	FIXED(3)
Mode:	FIXED(2)
Structure:	FIXED(1)
Byte Size:	FIXED(8)

III J 4 b (1) (b). Parameter field semantics

For detailed semantics of these fields see sections III A 4 c (1) - (4) above.

III J 4 b (2). Open

These parameters specify what file is to be transferred in what direction, the size of the file, and the record size, if applicable. The Security field is provided for file passwords or simply to revalidate the user's access privileges.

III J 4 b (2) (a). Parameter field syntax

Security:	VARIABLE(?)
Pathname:	VARIABLE(?)
Direction:	FIXED(2)
Position:	FIXED( )
Size:	FIXED( )

Record Size: FIXED( )

III J 4 b (2) (b). Parameter field semantics

For detailed semantics see sections III A 4 a and III A 4 b. Note: Files are closed implicitly by sending an Open request for a new file.

III J 4 b (3). Read

This request is sent by the UFAM to the SFAM to request that the amount of data specified by the parameter be transferred from the file. The read begins at the present position of the read pointer. When the operation is complete, the pointer is positioned one unit beyond the last unit of information read; i.e., at the next unit of information to be read.

III J 4 b (3) (a). Parameter field syntax

Amount: FIXED( )
Marker: VARIABLE( )

III J 4 b (3) (b). Parameter field semantics

The contents of the Amount field specifies the amount of data to be read from the file. The size of the field and the units represented by it are installation parameters.

The Marker field is used to synchronize the request with the data transferred by the TRANSMIT Command. This Marker and the Marker in the first TRANSMIT Command carrying data referred to by this Command should be the same.

III J 4 b (4). Write

This request is sent by the UFAM to the SFAM to request that the amount of data specified by the parameter be transferred to the file. The write begins at the present position of the write pointer. When the operation is complete, the write pointer is positioned one unit beyond the last unit written; i.e. at the next unit of information to be written.

III J 4 b (4) (a). Parameter field syntax

Amount: FIXED( )
Marker: VARIABLE( )

III J 4 b (4) (b). Parameter field semantics

The contents of the Amount field specifies the amount of data to be written to the file. The size of the field and the units specified by it are installation parameters.

The Marker field is used to synchronize the request with the data transferred by the TRANSMIT Command. This Marker and the Marker in the first TRANSMIT Command carrying data referred to by this Command should be the same.

III J 4 b (5). Set Read Pointer

This request is sent by the UFAM to the SFAM to request that the read pointer be set to the value indicated by the parameter. The next Read request will begin at this point.

III J 4 b (5) (a). Parameter Field Syntax

Position: FIXED( )

III J 4 b (5) (b). Parameter Field Semantics

The contents of this field indicate the new position of the read pointer. The size and units of the field are installation parameters. It is recommended that special values be provided to denote the beginning and end of the file.

III J 4 b (6). Set Write Pointer

This request is sent by the UFAM to the SFAM to request that the write pointer be set to the value indicated by the parameter. The next Write request will begin at this point.

III J 4 b (6) (a). Parameter field syntax

Position: FIXED( )

III J 4 b (6) (b). Parameter field semantics

The contents of this field indicate the new position of the write pointer. The size and units of the field are installation parameters. It is recommended that special values be provided to denote the beginning and end of the file.

III J 4 b (7). Abort

This request is sent by the receiving file access module to request that the current transfer of data be aborted. The sending FAM should immediately stop sending and await further commands.

III J 4 b (8). Open Network Connection

This request is sent by the UFAM to the SFAM to request that a network connection be opened according to the parameters.

III J 4 b (8) (a). Parameter field syntax

```
Connection parameters:                COMPLEX(8)
   Control-bits:                         FIXED(5)
      Init/Listen
      Socket/Channel
      Duplex/Simplex
      Relative/Absolute
      Unattachable/Attachable
   Host:                                 FIXED(32)
   Foreign Socket:                       FIXED(32)
   Messages:                             FIXED(16)
   Bits:                                 FIXED(32)
   Local Socket:                         FIXED(32)
   Timeout:                              FIXED(16)
   Byte Size:                            FIXED(8)
```

III J 4 b (8) (b). Parameter field semantics

For detailed semantics, see section III A 4 d.

III J 4 b (9). Set Marker Window

This request is sent by the sender of the data to set the width of the restart marker window. In protocols that use restart markers, this request is used to indicate how many markers can be sent without receiving a Marker Acknowledgement. The marker window should be set by the FAM sending the data. (Note: the use of the marker window mechanism is optional.)

III J 4 b (9) (a). Parameter field syntax

Window Width: FIXED( )

III J 4 b (9) (b). Parameter field semantics

The contents of this field indicate the width of the marker window. The size of this field and its units are installation parameters.

III J 4 b (10). Return Connection Status

No additional parameters need to be specified for this request.

III J 4 b (11). Marker Acknowledgement

This request is sent by the receiver of the data to inform the sender that all data sent before a specified marker have been written on the file. The receiver of this request should advance the left edge of the marker window to the position indicated by the parameter. (Acknowledgement of several markers is allowed.)

III J 4 b (11) (a). Parameter field syntax

Marker: FIXED( )

III J 4 b (11) (b). Parameter field semantics

This field contains the marker being acknowledged. If several markers are being acknowledged, this field contains the last marker in the sequence. The size of the field and the form of the restart markers are installation parameters.

III K. EXECUTE Response

III K 1. When sent

A file access module sends an EXECUTE Response when
a) it has completed the operation requested by an EXECUTE Command,
b) it has received an EXECUTE Command whose TEXT is in error, or
c) report on the status of the network connection.

III K 2. Action on receipt

When a FAM receives an EXECUTE Response, it should examine the TEXT to see whether the Response indicates an error. If no error is indicated, it may consider that the action requested by the previous EXECUTE Command was successful. If an error is indicated, appropriate error recovery action should be initiated. This action is idiosyncratic to the process.

III K 3. TEXT field syntax

Misc.-Result:	FIXED(4)
File Results:	FIXED(4)
Data Transfer Results:	FIXED(5)
Marker Results:	FIXED(3)
Ref-code:	FIXED(4)
Marker:	VARIABLE(?)
Connection Results:	COMPLEX(8)
Connection State	FIXED(4)
Host	FIXED(32)
Foreign Socket	FIXED(32)
Messages	FIXED(16)
Bits	FIXED(32)
Local Socket	FIXED(32)

III K 4. TEXT field semantics

III K 4 a. Misc.-Result

This field contains an encoding of the result of the request (in a previous EXECUTE Command) to which the EXECUTE Response is a reply. The codes are:

1	Success
2	Illegal code in request
4	TEXT too short
8	Illegal parameter combination

III K 4 b. File Results

This field contains an encoding of errors in the

file parameters. The default value is zero (no errors). The codes are:

0	Success
1	File not found
2	File name not allowed
3	File not presently available
4	Size too large
5	Record size too long
6	Position not valid
7	Direction not valid

III K 4 c. Data Transfer Results

This field contains an encoding of any errors in the Data Transfer Parameters. The codes are:

1	Improper type
2	Improper mode
4	Improper structure
8	Byte size not supported
16	Illegal parameter combination

III K 4 d. Marker Results

This field contains an encoding of any errors resulting from the handling of restart markers. The codes are:

0	Success
1	Marker window too wide
2	Marker window too narrow
3	Illegal marker
4	Marker outside window

III K 4 e. Ref-code

This field contains the same value as the Code field of the TEXT of the EXECUTE Command to which the EXECUTE Response is a reply.

III K 4 f. Marker

The Marker field is used to identify the EXECUTE Command for which this is the Response. This field should match the Marker field in the EXECUTE Command.

III K 4 g. Connection Results

This field is used in replies to Open Network Connection and Return Connection Status requests. It has the same meaning as its identically named and formatted counterpart in the TEXT of the BEGIN

Response, with the following exception:

III K 4 g (1). Connection-state

When the Response is to a Return Connection Status request, this field indicates the present state of the connection. The possible values of this field and their meanings are:

0	Closed
1	Open
2	Waiting for open to complete
3	Waiting for close to complete

IV. Description of the File Access PSP

Basic Operation. Although the primary purpose of the File Access PSP is to facilitate offloading FTP functions from a host, it serves the more general purpose of providing the host or front end access to the other machine's file system. This access is controlled by the UFAM. The UFAM first sends a BEGIN Command to the SFAM to instruct it to establish a session, to validate or identify the user, and to prepare for access to a particular file or network port. Once everything is in readiness data transfer may commence with the UFAM issuing reads and writes and transmitting data.

The File Access Process-to-Service Protocol associates a read pointer and a write pointer with an open file. The read and write pointers indicate the location where the next read or write operation will begin. After the read or write, the respective pointer is left pointing after the last unit of information read or written. Since the level of file addressing varies considerably from host to host the File Access PSP does not define the units represented by the read and write pointers. Implementations should choose units that allow efficient addressing and information transfer. Implementors should also provide a mechanism for the UFAM to specify the beginning or end of a file without actually knowing the address of the beginning or end.

When the transfer is complete, the UFAM may open another

file and transfer data to or from it. When the UFAM has finished its task it closes the channel by sending an END Command.

Coordination of Commands and data. In the FA-PSP, the operations to be performed (read, write, open, etc.) are transferred by TRANSMIT Commands. When reads and writes are used to access the file randomly, some way must be provided to match operations with their data, since EXECUTE Commands and Responses are sent out of band with TRANSMIT Commands.

Two methods may be used. The implementations in the host and front end may be designed so that an EXECUTE Command cannot be sent until the EXECUTE Response from the previous EXECUTE Command has been received. The SFAM would then withhold the EXECUTE Response until it had completed the operation. This method, however, does not allow overlapping requests and could significantly cut down the bandwidth. Alternatively, the Marker fields in the read and write EXECUTE Commands and in the TRANSMIT Commands can be used to tag the operations and the corresponding data. By this device the SFAM can determine what it is to do with the data by matching the marker in the EXECUTE Command with the data in the TRANSMIT Command. Similarly the UFAM can determine what operations have completed by matching the Marker field in an EXECUTE Response with the Markers for outstanding EXECUTE Commands.

Use of markers for restart. File Transfer Protocols

structured similarly to the File Access PSP do not require an explicit restart mechanism for transfers of complete files. If the host can determine the length of the file and an explicit address for the end of the file, then a restart can be accomplished by determining the address of the last data unit received and setting read and/or write pointers accordingly for the transfer. Since many FTP's do not provide the ability to address a file, the File Access PSP supports a fairly general restart mechanism that should be capable of encompassing most restart facilities found in FTP's.

There are two major aspects to this restart facility: 1) the use of markers associated with the data, and 2) a restart marker window (RMW). The protocol allows markers to be placed in the TRANSMIT Command with the data. When the data has been written on the file so that a system crash will not cause it to be lost, the FAM controlling the data sends an EXECUTE Command to the FAM controlling restart to acknowledge the marker. In order that one FAM not get too far ahead of the other, a restart marker window is provided. An EXECUTE Command variant is provided to set the width of the RMW; however, installations should define a default RMW width. This window works exactly like the ones found in low-level protocols such as TCP and the CYCLADES TS, except that in this case the window is used to control the flow of markers and only indirectly the flow of data. The sending FAM can continue to send data as long as the window is open. Each marker sent or received moves

the left edge of the window and each marker acknowledgement sent or received moves the right edge of the window. If the FAM controlling the data gets too far behind in acknowledging markers, the window will close and the transfer will stop until one or more markers are acknowledged. The File Access PSP does not specify the form of the markers or how they relate to the data or amount of data transferred. However, implementors would be wise to generate markers in a form such that, when a marker acknowledgement is received, it can be assumed to acknowledge all preceding markers. Also, implementors should try to define a marker form that makes it easy for the FAM controlling the transfer to be able to tell what markers came before the one currently under consideration and to what point in the file this restart point corresponds.

The RMW is useful for several reasons. It can ensure that if a failure occurs the amount that must be retransmitted is not too large. Also, if the format of markers is such that a table must be kept to map markers into the corresponding file address, the window restricts the table to a fixed, manageable size.

Use in offloading FTP. The File Access Process-to-Service Protocol is used in several different ways to offload file transfer protocols. Details of these schemes may be found in "Offloading ARPANET Protocols to a Front End," CAC Document 230. The most important distinction between the schemes is

whether the UFAM is in the host or in the front end.

If the UFAM is in the host, then the FA-PSP is used to offload network access for data transfer to and from remote hosts. If the data transfer process is in the front end, the FA-PSP is also used to offload data translation tasks. Unless the UFAM is attempting to access a file system in the front end, the file parameters of the BEGIN Command will not be used. Only the Data Connection Parameters will be present. The restart marker apparatus will only be present if marker bookkeeping and assignment is done in the host. These markers will be present, however, to implement the marker mechanism of the FTP rather than to offload it.

If the UFAM is in the front end, then the FA-PSP is being used to offload network access for data transfer, access to the host's file system, and possibly data translation and restart. The Connection Parameters will never occur in the BEGIN Command. The Data Parameters may occur if the data translation is done in the host. (However, translation will most likely be done in the front end.) The File Parameters will always be present. In this case, the FA-PSP can be used to offload more of the restart bookkeeping to the front end. Of course, markers and marker acknowledgements may be present merely to implement FTP restart, regardless of whether restart is offloaded.

V. Scenario

Let us consider how this protocol is used to transfer files between the host and front end. In this scenario, the Using File Access Module (UFAM) is in the front end. Another front-end module (e.g., the user or server FTP service module) has passed the UFAM the necessary information to initiate a transfer. To be specific, let us assume that the server FTP service module (SFSM) has just given the UFAM the user identification, pathname, and network port information for starting a transfer across the network. The UFAM then sends a BEGIN Command to the host to request access to the file:

<BEGIN><C><security=day,john><pathname=junk.note>
<direction =2>

The Serving File Access Module (SFAM) in the host successfully opens the file and sends a BEGIN Response:

<BEGIN><R><Status=0>

The UFAM is now ready to begin the transfer of the data. It notifies the SFSM, so it can send the user the FTP reply indicating begin of transfer, and sends a read to the SFAM:

<EXECUTE><C><code=2><Amount=ALL>

(This implementation has defined the string "ALL" to indicate that the entire file is to be read.) Data begins to flow immediately via the TRANSMIT Commands:

<TRANSMIT><C><special=0><the data>.

At some time, perhaps before the first TRANSMIT Command, the SFAM sends a response to the read:

<EXECUTE><R>

(No parameters are required since the read was successful.) Finally the last of the file is moved across:

<TRANSMIT><C><special=3><data>

The UFAM notifies the SFSM after the last message has been written to the net, so that the SFSM can send the FTP reply indicating the end of the transfer. The UFAM then waits for another request from the SFSM. However, the user is done and has sent an FTP BYE Command to the SFSM. The SFSM notifies the UFAM that the session is terminating and the UFAM sends an END Command indicating normal completion:

<END><C><cause=0>

The SFAM replies immediately with an END Response:

<END><R>.

VII. References

1. Day, J. "Offloading ARPANET Protocols to a Front End," CAC Document No. 230, Center for Advanced Computation, University of Illinois at Urbana-Champaign, 1977.

2. Neigus, N. "The File Transfer Protocol," ARPANET RFC 542, 1973.

3. Neigus, N.; Pogran, K.; and Postel, J. "A New Schema for FTP Reply Codes," ARPANET RFC 640, 1974.

4. Schicker, P.; Duenki, A.; and Baechi, W. "Bulk Transfer Facility," INWG Protocol Note #31. EIN/ZHR/75/20, 1976.

Appendix 3

ARPANET Host-Host Process-to-Service Protocol Specification

CAC Technical Memorandum 80

by

John Day
Gary Grossman

ARPANET Host-Host

Process-to-Service Protocol

Specification

I. Service Code Number

1

II. Description of the Service

This document is a process-to-service protocol specification as defined in the Host-to-Front-End Protocol (HFP) Specification [ARPANET RFC 710]. It specifies a process-to-service protocol for providing ARPANET Host-Host Protocol and Initial Connection Protocol services to a process through the HFP.

The Host-Host Protocol is the basic inter-process communication protocol for the ARPANET [ARPANET NIC Document 8246]. The program which implements it in each host is the Network Control Program (NCP). The service described here provides an interface, through the HFP, between a process in a host and an NCP in a front end. This enables the process to establish and use ARPANET connections.

The Initial Connection Protocol (ICP) is the mechanism in the ARPANET which enables a process in one host to connect to a multi-user facility, such as the virtual terminal server or the file transfer server, in another host [ARPANET NIC Documents 7101, 7155, 7103]. The service described here enables a process

in a host to establish ARPANET connections via the ICP.

The ARPANET Host-Host process-to-service protocol uses HFP Messages to carry information between the process and the service module. A brief summary of the function of specific Message types follows.

<u>BEGIN Command</u>
from the host process
requests that a connection be established.

<u>BEGIN Response</u>
from the service module
a) confirms the establishment of the connection or
b) reports the reason for failure.

<u>END Command</u>
from the process
requests the closing of the connection.
from the service module
reports the closing of the connection by the foreign host.

<u>END Response</u>
from the process
acknowledges the closing of the connection.
from the service module
acknowledges the closing of the connection.

<u>TRANSMIT Command</u>
from the process
carries data to the service module for transmission to the foreign host over the ARPANET.
from the service module
carries data received over the ARPANET to the process.

<u>EXECUTE Command</u>
from the process
requests that the service module
a) send an ARPANET Host-Host INR command referencing the connection,
b) alter flow control parameters for the connection, or
c) report the status of the connection.

<u>EXECUTE Response</u>

from the service module

a) acknowledges the completion of actions requested by the process via EXECUTE Commands and

b) reports the status of the connection.

Security and protection problems are posed by the way that local sockets are used in the ARPANET protocols. If processes were permitted to use any socket number without restriction, a process could 'masquerade' as another (e.g., the logger) and gain access to privileged information (e.g., passwords). For this reason, two modes of access to the local socket name space are provided. The "relative" mode restricts the process to a unique, distinct subset of the local socket name space. This mode requires no special privilege. The "absolute" mode allows the process to access any local socket number. Its use requires the process to have special privileges. The modes are distinguished by the Relative/Absolute bit in the TEXT of the BEGIN Command. (See sections III A 4 b (1) (d) and III A 4 b (6).)

Some front ends may be so structured that it is possible for service modules to communicate with each other. This facility could be used to permit service modules such as Telnet to communicate with the ARPANET host-host service module and use network connections which the process had already separately established. A process could request the host-host service module to connect to a remote host. Then the process could request the Telnet service to "attach" to the already existing logical channel associated with the host-host service module and thus use the already existing network connection. The

"Unattachable/Attachable" bit in the TEXT of the BEGIN Command is used to enable this feature for a given logical channel. See section III A 4 b (1) (e).

The detailed specification of the use of HFP Messages follows.

III. Message Use

III A. BEGIN Command

III A 1. When sent

A BEGIN Command is sent by the process to request that the service module attempt to establish an ARPANET connection.

III A 2. Action on receipt

When the service module receives the BEGIN Command, it will attempt to establish the connection according to the parameters in TEXT.

III A 3. TEXT field syntax

```
Security:                  VARIABLE(?)
Establishment:             COMPLEX(8)
   Control-bits:              FIXED(5)
      Init/Listen
      ICP/Direct
      Duplex/Simplex
      Relative/Absolute
      Unattachable/Attachable
   Host:                      FIXED(32)
   Foreign Socket:            FIXED(32)
   Messages:                  FIXED(16)
   Bits:                      FIXED(32)
   Local Socket:              FIXED(32)
   Timeout:                   FIXED(16)
   Byte Size:                 FIXED(8)
```

III A 4. TEXT field semantics

III A 4 a. Security

This field can be used to validate the process' access privileges to network resources. It is particularly important to limit access to network security objects such as local socket 1. The substructure and content of this field are installation dependent.

NOTE: At this writing, not enough is known about security in networks and front ends to make any statement about the utility of validation at this point in the system. This field is included in case some systems require validation at this point.

III A 4 b. Establishment

This set of parameters governs the establishment of the network connection.

III A 4 b (1). Control-bits

The bits in this field govern the kind of connection to be established, the method of its establishment, and the interpretation of the parameters.

III A 4 b (1) (a). Init/Listen

When this bit is zero (default), the connection is to be actively initiated by the NCP. When this bit is one, the connection is to be passively "listened" for by the NCP. The use of this bit in specifying connection sockets is detailed in table 1.

III A 4 b (1) (b). ICP/Direct

When this bit is zero (default), the connection is to be established via the Initial Connection Protocol. When this bit is one, the connection is to be established directly to the specified foreign socket. The use of this bit in specifying connection sockets is detailed in table 1.

III A 4 b (1) (c). Duplex/Simplex

This bit is relevant only if the ICP/Direct bit is one. When this bit is zero (default), the connection is to be full duplex and is therefore to consist of two ARPANET connections (one in each direction). When this bit is one, the connection is to be unidirectional according to the gender of the socket numbers supplied in the Foreign Socket and/or Local Socket fields. The use of this bit in specifying connection sockets is detailed in table 1.

III A 4 b (1) (d). Relative/Absolute

This field affects the value of the socket for the local connection. See III A 4 b (6) for details.

III A 4 b (1) (e). Unattachable/Attachable

When this bit is one, the host-host service

module is requested to make the logical channel available for attachment by other service modules within the front end (such as Telnet or a data transformation service). If this bit is zero (default), the host-host service module is not requested to make the logical channel available for attachment within the front end.

III A 4 b (2). Host

This field specifies the network host to which the connection is to be made. A value of zero in this field indicates that the connection may be established with any host on the network. A zero value is valid only if the Init/Listen bit is one (Listen).

The Host field is parsed as:

Network:	FIXED(8)
Imp:	FIXED(16)
Host-at-Imp:	FIXED(8)

III A 4 b (3). Foreign Socket

This field is used as follows to construct the effective foreign socket (e.f.s.) for making the connection.

Case 1. Foreign Socket field = zero; Init/Listen bit = one. Foreign host chooses the e.f.s.

Case 2. Foreign Socket field = zero; Init/Listen bit = zero. The e.f.s. is one.

Case 3. Foreign Socket field ≠ zero. The e.f.s. is the contents of the Foreign Socket field.

The default value of this field is zero. The effective foreign socket is used in various ways, depending upon the values of certain of the control bits, to effect the connection. See table 1 for details.

III A 4 b (4). Messages

This field and the Bits field allow the process to indicate to the service module the flow rate desirable on the connection. The service module may use this information in any way its implementors deem desirable.

This field contains the number of ARPANET Host-

Host messages the process suggests be allocated on the connection. If the value of this field is zero (default), the service module chooses the number of messages on its own.

III A 4 b (5). Bits

This field contains the number of bits the process suggests be allocated for the connection. If the value of this field is zero (default), the service module chooses the number of bits on its own.

III A 4 b (6). Local Socket

This field is used as follows to construct the effective local socket (e.l.s.) for making the connection.

Case 1. Relative/Absolute bit = zero, Local Socket field ≠ zero. In this case, the contents of the Group field in the HEADER will be concatenated with the low-order 20 bits of the Local Socket field to form the e.l.s.

Case 2. Relative/Absolute bit = zero, Local Socket field = zero. In this case, the service will choose a number whose high order 12 bits are equal to the Group field in the HEADER.

Case 3. Relative/Absolute bit = one. In this case the effective local socket is equal to the contents of the Local Socket field.

The effective local socket is used in various ways, depending upon the values of certain of the control bits, to effect the connection. See table 1 for details.

III A 4 b (7). Timeout

This field contains the maximum length of time, in seconds, that the service module is to attempt to establish the connection (in the absence of an error or exceptional condition). If the service module has been unable to establish the connection within this time period, it is to abandon the attempt. If this field contains zero (default), the time period is to be infinite.

III A 4 b (8). Byte Size

This field contains the byte size for data

transfer on the connection. If this field contains zero (default), the byte size will be 8 bits.

Table 1

Control Bits		Foreign Sockets for the connection	Local Sockets for the connection
ICP	Init	The e.f.s. will be used as the contact socket for the connection. The data sockets will be chosen by the foreign host.	e.l.s. +2 and e.l.s. +3
	Listen	e.f.s. +2 and e.f.s. +3	The e.l.s. will be used as the contact socket for the connection. The data sockets will be chosen by the service as in Case 2 of III A 4 b (6).
Direct	Duplex	e.f.s. and e.f.s. +1	e.l.s. and e.l.s. + 1
	Simplex	e.f.s.	e.l.s.

III B. BEGIN Response

III B 1. When sent

A BEGIN Response is sent by the service module to report the outcome of an attempt to establish a connection requested by a BEGIN Command. The Status field in the HEADER will indicate whether the connection attempt was successful or unsuccessful.

III B 2. Action on receipt

When the process receives a BEGIN Response indicating a successful connection, it may proceed to send and receive data on the connection. When the process receives a BEGIN Response indicating an unsuccessful connection, it should perform the appropriate error recovery.

III B 3. TEXT field syntax

```
Connection-info:            COMPLEX(?)
   Connection-state:           FIXED(4)
   Host:                       FIXED(32)
   Foreign Socket:             FIXED(32)
   Messages:                   FIXED(16)
   Bits:                       FIXED(32)
   Local Socket:               FIXED(32)
   Byte Size:                  FIXED(8)
```

III B 4. TEXT field semantics

III B 4 a. Connection-state

If the connection attempt was successful (as indicated by the Status field in the HEADER), this field will contain the value 1 which will be compatible with the "open" state value defined for this field in the TEXT of the EXECUTE Response (which see).

If the connection attempt was unsuccessful, this field contains an encoded reason for the failure. The codes are:

0	Illegal control bit combination
1	Invalid local socket
2	Invalid host
3	Byte size not supported by front end
4	Byte size not supported by foreign host
5	Connection attempt timeout
6	Network not available
7	Foreign host dead
8	Connection refused
9	Front end terminating service

III B 4 b. Host

If the connection attempt was successful, this field contains the ARPANET host address of the host to which the connection was established.

III B 4 c. Foreign Socket

If the connection attempt was successful, this field contains the socket number at the foreign host to which the connection was established.

III B 4 d. Messages and Bits

If the connection attempt was successful, these fields contain the flow control parameters for the connection at the time the BEGIN Response was generated.

III B 4 e. Local Socket

If the connection attempt was successful, this field contains the local socket number for the connection.

III B 4 f. Byte Size

If the connection attempt was successful, this field contains the byte size, in bits, for the connection.

III C. END Command

III C 1. When sent

III C 1 a. By the process

An END Command is sent by the process to request the service module to close the network connection.

III C 1 b. By the service module

An END Command is sent by the service module to notify the process that the connection has been closed by the foreign host or by a failure of the foreign host or the network.

III C 2. Action on receipt

III C 2 a. By the process

When the process receives an END Command, it should consider the connection closed. The process should send an END Response and take appropriate recovery action.

III C 2 b. By the service module

When the service module receives an END Command, it should allow data queued for transmission to the network to drain, unless Flush away (bit 1) is set in the Control field. It should then initiate the process of closing the connection. When the connection is closed, the service module should send an END Response.

III C 3. TEXT field syntax

Cause: FIXED(3)

III C 4. TEXT field semantics

III C 4 a. Cause

This field contains an encoded reason for sending the END Command. The codes are:

0	Normal completion (default)
1	Network/process failure
2	Foreign host failure
3	User requested termination
4	Front end terminating service

III D. END Response

III D 1. When sent

III D 1 a. By the process

The END Response is sent by the process to acknowledge the receipt of an END Command and complete the termination of the associated logical channel.

III D 1 b. By the service module

The END Response is sent by the service module to acknowledge receipt of an END Command, notify the process that the connection has been closed as requested, and terminate the logical channel.

III D 2. Action on receipt

III D 2 a. By the process

When the process receives the END Response, it should consider the ARPANET connection to be closed and the HFP logical channel to be terminated.

III D 2 b. By the service module

When the service module receives the END Response, it should consider the logical channel to be terminated.

III D 3. TEXT field syntax

The TEXT field in the END Response is not used in this process-to-service protocol.

III E. TRANSMIT Command

III E 1. When sent

III E 1 a. By the process

The TRANSMIT Command is sent by the process to pass data to the service module for transmission over the network to the foreign host.

III E 1 b. By the service module

The TRANSMIT Command is sent by the service module to pass data that has arrived over the network from the foreign host to the process.

III E 2. Action on receipt

III E 2 a. By the process

When the process receives a TRANSMIT Command, it will treat the contents of the TEXT as data to be processed.

III E 2 b. By the service module

When the service module receives a TRANSMIT Command, it will treat the contents of the TEXT as data to be transmitted over the network.

III E 3. TEXT field syntax

TEXT contains the data to be transferred by the TRANSMIT Command.

III E 4. TEXT field semantics

The data in the TEXT have no other semantics than those described in sections III E 1 and III E 2.

III F. TRANSMIT Response

The TRANSMIT Response is used as described in the HFP specification.

III G. SIGNAL Command

The SIGNAL Command is not used in the process-to-service protocol. SIGNAL Commands may be generated by process or service to request that the CPM carry out the appropriate action on the logical channel. (See HFP specification.) Thus, the effect of the SIGNAL Command is restricted to the logical channel (HFP) level.

III H. SIGNAL Response

The SIGNAL Response is not used by either process or service.

III J. EXECUTE Command

III J 1. When sent

The process sends an EXECUTE Command to request that the service module

a) send an ARPANET Host-Host INR command,
b) alter the suggested allocation on the connection, or
c) return a description of the status of the connection.

The service module does not send EXECUTE Commands for this protocol.

III J 2. Action on receipt

When the service module receives an EXECUTE Command, it will decode the TEXT and act upon it.

III J 3. TEXT field syntax

```
Code:           FIXED(2)
Messages:       FIXED(16)
Bits:           FIXED(32)
```

III J 4. TEXT field semantics

III J 4 a. Code

This field contains an encoding of the type of request. The codes are:

0	Send INR
1	Alter suggested allocation
2	Return connection status

The Messages and Bits fields need be present only if the Code field contains the value 1 (Alter suggested allocation).

III J 4 b. Messages

This field is valid only if the request is "Alter suggested allocation" (Code = 1). It contains the number of ARPANET messages the process suggests be allocated on the connection. If the value of this field is zero (default), the service module chooses the number of messages on its own.

III J 4 c. Bits

This field is valid only if the request is "Alter suggested allocation" (Code = 1). It contains the number

of bits the process suggests be allocated for the connection. If the value of this field is zero (default), the service module chooses the number of bits on its own.

III K. EXECUTE Response

III K 1. When sent

The service module sends an EXECUTE Response when
a) it has completed the operation requested by an EXECUTE Command or
b) it has received an EXECUTE Command whose TEXT is in error.

The service module does not send EXECUTE Commands for this process-to-service protocol. Therefore, the process does not send EXECUTE Responses.

III K 2. Action on receipt

When the process receives an EXECUTE Response, it should examine the TEXT to see whether the response indicates an error. If no error is indicated, it may consider that the action requested by a previous EXECUTE Command was successful. If an error is indicated, appropriate error recovery action should be initiated. This action is idiosyncratic to the process.

III K 3. TEXT field syntax

Result:	FIXED(2)
Ref-code:	FIXED(2)
Connection-info:	COMPLEX(?)
Connection-state:	FIXED(2)
Host:	FIXED(32)
Foreign Socket:	FIXED(32)
Messages:	FIXED(16)
Bits:	FIXED(32)
Local Socket:	FIXED(32)
Byte Size:	FIXED(8)

III K 4. TEXT field semantics

III K 4 a. Result

This field contains an encoding of the result of the request (in a previous EXECUTE Command) to which the EXECUTE Response is a reply. The codes are:

0	Successful
1	Illegal code in request
2	TEXT too short

The "TEXT too short" error can occur only if the request was "Alter suggested allocation".

III K 4 b. Ref-code

This field contains the same value as the Code field of the TEXT of the EXECUTE Command to which the EXECUTE Response is a reply.

III K 4 c. Connection-info

This field has the same meaning as its identically named and formatted counterpart in the TEXT of the BEGIN Response, with the following exception:

III K 4 c (1). Connection-state

This field indicates the present state of the connection. The possible values of this field and their meanings are:

0	Closed
1	Open
2	Waiting for open to complete
3	Waiting for close to complete

IV. Scenario

Let us consider how this PSP can be used to remotely control a Network Control Program in the front end. We will assume that there is a program in the host that wishes to open a Telnet connection to remote host 12. A BEGIN Command is sent to the front end:

<BEGIN><C><host=12><msgs=15><bits=12000>

Because the defaults for the HH-PSP favor initiating a Telnet connection (i.e., a duplex connection established by an ICP to socket 1) the BEGIN Command does not have to specify many of the parameters. The HH Service in the front end opens the connection and sends a BEGIN Response to the host:

<BEGIN><R><STATUS=0><state=1><host=12><Fgnskt=5239>
<msgs=12><bits=12000>

indicating that the connection attempt was successful. The program now sends and receives via TRANSMIT Commands.

At some time later the program in the host receives an END Command:

<END><C><cause=Foreign host failure>

This indicates that the network connection has been closed because the remote host crashed. The front end wishes to terminate the service. The program immediately sends an END Response:

<END><R>

The channel is now closed. At some later time the program may attempt to re-establish the connection.

V. References

1. McKenzie, A.A. "Host/Host Protocol for the ARPA Network," NIC #7147, 1972. Also in the ARPANET Protocol Handbook.

2. Postel, J. "Official Initial Connection Protocol," NIC #7101, 1971. Also in the ARPANET Protocol Handbook.

Appendix 4

Network Virtual Terminal Process-to-Service Protocol Specification

CAC Technical Memorandum 103

by

John Day

Network Virtual Terminal

Process-to-Service Protocol

Specification

I. Service Code Number

5

II. Description of the Service

This document is a process-to-service protocol specification as defined in the Host-to-Front-End Protocol (HFP) Specification [ARPANET RFC 710]. It describes a protocol for offloading a network virtual terminal protocol (e.g., Telnet for the ARPANET) to a front end. This protocol allows some flexibility in the degree of offloading that may be achieved. Although the protocol is applicable to a general virtual terminal service, the discussion below is in terms of the ARPANET Telnet Protocol, which is currently the only such protocol widely used.

The functions of the typical network virtual terminal implementation are:

1. manipulating network connections,
2. negotiating Telnet options,
3. mapping between local terminal representations and network virtual terminal representations,
4. transmitting data over connections,
5. handling special control functions, and

6. interfacing the remote network connections so that they appear to the host to be local terminals.

The offloading scheme represented by this model places manipulating connections and transmitting data over the net in the front end. The interfacing of remote terminals so that they appear as local terminals must be done in the host. The scheme is flexible in that the remaining functions (option negotiation and translation between local and network representations) may be implemented in either the front end or the host, depending on local installation constraints. Typically the translation will be done in the front end. Section IV below discusses the offloading of Telnet options.

The only support this service requires in the front end is access to an ARPANET NCP. In the host it is necessary for the residual part of the Telnet implementation to be able to make its logical channels appear as if they were terminals connected to the host. This may require system modification in some cases.

There are several possible ways in which service could be initiated and controlled. The scheme described in this protocol appears to be the simplest and most flexible. In this scheme, BEGIN Commands only go from the host to the front end. When the front end service module receives a BEGIN Command, it either does a listen on the contact socket or it sets up a connection as requested. It sends a BEGIN Response to the host when a

connection is established. The host process must generate a distinct BEGIN Command for each connection it is willing to accept. Thus, the host is able to limit the number of Telnet users by withholding BEGIN Commands. The front end may also limit demand on its resources by refusing BEGIN Commands from the host.

The network virtual terminal process-to-service protocol uses HFP Messages to carry information between the residual part of Telnet in the host (the "process") and the network virtual terminal service module (the "service") in the front end. A brief summary of the specific Message types follows.

<u>BEGIN Command</u>
is used to initiate connections.

<u>BEGIN Response</u>
is used to confirm the establishment of a connection, or to refuse a BEGIN Command.

<u>END Command</u>
is used to terminate a connection.

<u>END Response</u>
is used to confirm the termination.

<u>TRANSMIT Command</u>
is used to transfer data (including Telnet option negotiations and control functions) to and from the Telnet connection.

<u>TRANSMIT Response</u>
is used to acknowledge data transfer.

<u>EXECUTE COMMAND</u>
is used by the process to request the service module
a) to send a Telnet control function,
b) to modify the suggested allocation,
c) to negotiate Telnet options, or
d) to report the status of the connection.

is used by the service to pass to the process out-of-band Telnet control functions (e.g. IP, AO, and Break).

<u>EXECUTE Response</u>
a) is used to acknowledge the success or failure of the action requested by the process via the EXECUTE Command, and
b) if requested reports on the status of the connection.

Systems which can easily set terminal parameters may offload many functions using the EXECUTE Command. For such systems adaptations of this process-to-service protocol could define requests in a form that would allow the host to be ignorant of the actual negotiation mechanism. For instance, an adaptation could define an EXECUTE Command request to indicate that a certain system primitive was to be called and given such and such parameters. Since not all systems can do this, and the means available to the ones that can differ considerably, we have left the specification of such features for the adaptation descriptions.

This process-to-service protocol requires the implementation in the front end of the following additional protocols:

ARPANET Host-Host Protocol

ARPANET Initial Connection Protocol

ARPANET Telnet Protocol

The details of the use of HFP Messages are given in the following section.

III. Message Use

III A. BEGIN Command

III A 1. When sent

A BEGIN Command is sent by the host process to indicate that it wishes to establish a Telnet connection. The host process sends a BEGIN Command for each Telnet connection it will accept.

III A 2. Action on receipt

When the service module receives a BEGIN Command, it attempts to set up a connection as specified by the parameters in the TEXT. The service module may refuse the BEGIN Command by sending an appropriate BEGIN Response.

III A 3. TEXT field syntax

```
Security:                                   VARIABLE(?)
Establishment:                              COMPLEX(8)
   Control-bits:                               FIXED(5)
      Init/Listen
      Socket/Channel
      ICP/Direct
      Relative/Absolute
      Unattachable/Attachable
   Host:                                       FIXED(32)
   Foreign Socket:                             FIXED(32)
   Messages:                                   FIXED(16)
   Bits:                                       FIXED(32)
   Local Socket:                               FIXED(32)
   Timeout:                                    FIXED(16)
```

III A 4. TEXT field semantics

III A 4 a. Security

This field may be used by the host to validate the access privileges of the sender of the BEGIN Command. The default value of the field is empty. Note: At this time, very little work has been done on the organization of security for networks or these kinds of systems. It is not clear where access control should be placed. This field has been included in case some systems require validation at this point.

III A 4 b. Establishment

This set of parameters describes the kind of connection to be associated with this channel. The default values of the establishment parameters are

chosen to favor the Server Telnet service. This means that a BEGIN Command specifying only defaults would cause the front end to listen for an ICP connection on the standard Telnet contact socket (socket 1) from any host for an indefinite period of time. The default values of the fields are set at zero; the default value may be indicated by the absence of the field. Thus, a BEGIN Command sent to the front end for a server Telnet session may have an empty TEXT field.

III A 4 b (1). Control-bits

The bits in this field govern the kind of connection to be established and the interpretation of the parameters.

III A 4 b (1) (a). Listen/Init

When this bit is zero (default), the connection is to be passively listened for. When this bit is one, the connection is to be actively initiated. The use of this bit in specifying connection sockets is detailed in table 1.

III A 4 b (1) (b). Socket/Channel

If this bit is zero, the logical channel from the process is to be established to a new ARPANET connection. If this bit is one, the logical channel from the process is to be attached to an already existing HFP channel (e.g. a channel already established to the host-host service module and therefore providing access to an existing network connection.) The default value is zero.

III A 4 b (1) (c). ICP/Direct

This bit is relevant only when a new network connection is requested. (Socket/Channel bit is zero.) When this bit is zero (default), the connection is to be established via the Initial Connection Protocol. When this bit is one, the connection is to be established directly. The use of this bit in specifying connection sockets is detailed in table 1.

III A 4 b (1) (d). Relative/Absolute

This field affects the value of the local sockets for the connection. See III A 4 b (6) for details. The default case is Relative (value zero).

III A 4 b (1) (e). Unattachable/Attachable

When this bit is one, the network terminal service module is requested to make the logical channel available for attachment by other service modules within the front end (such as a data transformation service). Otherwise this bit is zero (default). See CAC Technical Memorandum No. 80, p. 3, for a discussion of this feature.

III A 4 b (2). Host

This field specifies the network hosts from which (or to which) the Telnet connection is to be established. The default value is zero, indicating that a connection from any host will be accepted. (The default value is legal only if the Listen/Init bit is zero (Listen), otherwise a host must be specified.)

The host field is parsed as:

Network:	FIXED(8)
IMP:	FIXED(16)
Host-at-IMP:	FIXED(8)

III A 4 b (3). Foreign Socket

This field is used as follows to construct the effective foreign socket (e.f.s.) for making the connection.

Case 1. Foreign Socket field = zero; Listen/Init bit = zero. Foreign host chooses the e.f.s.

Case 2. Foreign Socket field = one; Listen/Init bit = zero. The e.f.s. is one.

Case 3. Foreign Socket field ≠ zero. The e.f.s. is the contents of the Foreign Socket field.

The default value of this field is zero. The effective foreign socket is used in various ways, depending upon the values of certain of the control bits, to effect the connection. See table 1 for details.

III A 4 b (4). Messages

This field and the Bits field allow the process to indicate to the service module the flow rate desirable on the connection. The service module may use this information in any way its implementors deem

desirable.

This field contains the number of Host-Host messages the process suggests be allocated on the connection. If the value of this field is zero (default), the service module chooses the number of messages on its own.

III A 4 b (5). Bits

This field contains the number of bits the process suggests be allocated for the connection. If the value of this field is zero (default), the service module chooses the number of bits on its own.

III A 4 b (6). Local Socket or Logical Channel

If the Socket/Channel bit is one, this field contains the channel Group and Member values specifying the HFP channel to which connection is to be made. The format of this field is then:

```
<not used>: FIXED(4)
 Group:     FIXED(12)
 Member:    FIXED(16)
```

If the Socket/Channel bit is zero, this field is used as follows to construct the <u>effective local socket</u> (e.l.s.) for making the connection.

Case 1. Relative/Absolute bit = zero, Local Socket field ≠ zero. In this case, the contents of the Group field in the HEADER will be concatenated with the low-order 20 bits of the Local Socket field to form the e.l.s.

Case 2. Relative/Absolute = zero, Local Socket field = zero. In this case, the service will choose a number whose high order 12 bits are equal to the Group field in the HEADER.

Case 3. Relative/Absolute = one. In this case the effective local socket is equal to the contents of the Local Socket field.

The effective local socket is used in various ways, depending upon the values of certain of the control bits, to effect the connection. See table 1 for details.

III A 4 b (7). Timeout

This field contains the maximum length of time, in seconds, that the service module is to attempt to

establish the connection (in the absence of an error or exceptional condition). If the service module has been unable to establish the connection within this time period, it is to abandon the attempt. If this field contains zero (default), the time period is to be infinite.

Table 1

Control Bits		Foreign Sockets for the connection	Local Sockets for the connection
ICP	Init	The e.f.s. will be used as the contact socket for the connection. The data sockets will be chosen by the foreign host.	e.l.s. +2 and e.l.s. +3
	Listen	e.f.s. +2 and e.f.s. +3	The e.l.s. will be used as the contact socket for the connection. The data sockets will be chosen by the service as in Case 2 of III A 4 b (6).
Direct		e.f.s. and e.f.s. +1	e.l.s. and e.l.s. + 1

III B. BEGIN Response

III B 1. When sent

The BEGIN Response is sent by the service module to indicate that a successful network connection has been established, or that the BEGIN Command has been refused.

III B 2. Action on receipt

When the BEGIN Response is received, the host process should determine whether a successful connection was made. If so, it may send or receive data. If not, it may be appropriate to invoke an error recovery procedure.

III B 3. TEXT field syntax

```
Security:                        VARIABLE(?)
Connection-info:                 COMPLEX(?)
   Connection-state:                FIXED(4)
   Host:                            FIXED(32)
   Foreign Socket:                  FIXED(32)
   Messages:                        FIXED(16)
   Bits:                            FIXED(32)
   Local Socket:                    FIXED(32)
```

III B 4. TEXT field semantics

III B 4 a. Security

This field may be used to allow the front end to validate the access privileges of the user (in the case where the front end performs some user authentication tasks) and to pass the information along to the host. This field could be particularly important for monitoring access by terminals directly connected to the front end.

III B 4 b. Connection-info

This field describes the connection associated with this channel.

III B 4 b (1). Connection-state

This field indicates the success or failure of the BEGIN Command. The value zero indicates success while any non-zero value indicates failure. The particular value of the non-zero field gives the reason for failure. The possible values of this field and their meanings are:

0	Success
1	Illegal connection type
2	Invalid local socket
3	Unknown host
4	Improper access
5	Front end terminating service
6	Connection attempt timeout
7	Network not available
8	Foreign host dead
9	Connection refused by foreign host

Others may be defined later, or in the adaptation descriptions.

III B 4 b (2). Host

If the connection attempt was successful, this field contains the ARPANET host address of the host to which the connection was established. (See III A 4 b (2).)

III B 4 b (3). Foreign Socket

If the connection attempt was successful, this field contains the socket number at the foreign host to which the connection was established.

III B 4 b (4). Messages and Bits

If the connection attempt was successful, these fields contain the flow control parameters for the connection at the time the BEGIN Response was generated.

III B 4 b (5). Local Socket

If the connection attempt was successful, this field contains the local socket number for the connection.

III C. END Command

III C 1. When sent

III C 1 a. By the process

The END Command may be sent by either the process or the service to terminate a Telnet connection. If sent by the process, it usually indicates that the host has requested that the connection be closed.

III C 1 b. By the service module

The service module may send an END Command because the foreign host has closed the connection in response to a user request, or because some failure condition has occurred that requires that the connection be aborted.

III C 2. Action on receipt

III C 2 a. By the process

When the process receives an END Command, it should acknowledge it as promptly as possible by sending an END Response.

III C 2 b. By the service module

When the service module receives an END Command, it should close the Telnet connection. Once the connection is closed, it should send the END Response to the process.

III C 3. TEXT field syntax

Status: FIXED (3)

III C 4.TEXT field semantics

III C 4 a. Status

This field indicates the cause of the termination of the TELNET connection. The values of the field and their associated meanings are:

0	Normal close by foreign host
1	Network failure
2	Foreign host failure
3	User requested close
4	Improper Access
5	Front end terminating service

Other causes for failure may be defined at a later date.

The default value of the field is zero.

III D. END Response

III D 1. When sent

The END Response is sent by either the process or the service module to acknowledge that an END Command has been received and proper action taken.

III D 2. Action on receipt

When either the process or the service receives an END Response, it should assume that all dialogue associated with this logical channel has completed, and it should ignore any other Commands on this channel except a BEGIN Command.

III D 3. TEXT field syntax

The TEXT field in the END Response is not used in this protocol.

III E. TRANSMIT Command

The description below assumes that a Go-Ahead facility is needed for the handling of half duplex terminals. Not all installations will need this facility; those that do not should set the Go-Ahead/More-Data Control bit to zero and ignore the specification of how this bit is interpreted.

III E 1. When sent

The TRANSMIT Command may be sent by either side to transfer data. Data may be sent only when the connection is in the ESTABLISHED state and when the flow control discipline permits. (See HFP Specification.)

III E 2. Action on receipt

III E 2 a. By the process

When the process receives a TRANSMIT Command, it will treat the contents of the Data field of the TEXT as data to be processed.

III E 2 b. By the service module

When the service module receives a TRANSMIT Command, it will treat the contents of the Data field of the TEXT as data to be transmitted over the Telnet connection.

III E 3. TEXT field syntax

The TEXT field contains one or more instances of the following structure.

```
Control-bits:                 FIXED(2)
    Text/Control
    Go-Ahead/More-Data
Data:                         VARIABLE(?)
```

III E 4. Text field semantics

III E 4 a. Control bits

The Control bits indicate the nature and the mode of the information in the associated Data field being passed to the process or service.

III E 4 a (1). Text/Control

When this bit is zero (default), it indicates that the Data field contains text to be transmitted. When this bit is one, it indicates that the Data

field contains a specific Telnet option or control function.

III E 4 a (2). Go-Ahead/More-Data

When this bit is zero (default), it specifies that the sender has transmitted all the data it wishes and the receiver may now "turn the line around" and send data. It is convenient for the host process if the service makes each Data field of the TEXT correspond to one line of input from the terminal, including the end-of-line character. (This may be agreed on by the service and process.) In this way, only one of the processes must be bothered with looking for end-of-line sequences. If this bit is one, there is more data to follow.

III E 4 b. Data

If the Text/Control bit is one, the Data field will contain the text of a Telnet option negotiation or control function. If the Text/Control bit is zero (default), the Data field contains text which is to be transmitted or has been received on the Telnet connection.

III F. TRANSMIT Response

The TRANSMIT Response is used as described in the HFP specification.

III G. SIGNAL Command

The SIGNAL Command is not used in the process-to-service protocol. SIGNAL Commands may be generated by process or service to request that the channel protocol module (CPM) carry out the appropriate action on the logical channel. (See HFP specification.) Thus, the effect of the SIGNAL Command is restricted to the logical channel (HFP) level.

III H. SIGNAL Response

The SIGNAL Response is not used by either process or service.

III J. EXECUTE Command

III J 1. When sent

By the process

The process sends an EXECUTE Command to request that the service module

a) send a Telnet control function;
b) modify the suggested allocation;
c) negotiate a Telnet option; or
d) return the status of the connection.

By the service

The service sends an EXECUTE Command to the process to notify it that certain Telnet control functions have been received.

III J 2. Action on receipt

When the service module or process receives an EXECUTE Command, it will decode it and perform the action indicated. (See below.)

III J 3. TEXT field syntax

Code:	FIXED(2)
Parameters:	VARIABLE(?)

III J 4 TEXT field semantics

III J 4 a. Code

This field contains an encoding of the type of request. The codes are:

0	Send control function
1	Modify suggested allocation
2	Negotiate Telnet option
3	Status

III J 4 b. Parameters

This field, if present, contains any parameters necessary to perform the request indicated in the code field. A discussion of the requests and their associated parameters follows.

III J 4 c. Send Control Function

III J 4 c (1). When sent

By the process

This request is sent to the service to request the service module to send one of the out-of-band Telnet control functions (IP, AO, AYT, or Break) to the foreign host.

By the service

This request is sent by the service to notify the process that an out-of-band Telnet control function has been received from the foreign host.

III J 4 c (2). Action on receipt

By the process

When the process receives a control function request, it should take whatever action is required according to the semantics defined by the Telnet protocol. The only exception to this is the Abort Output control function. When the process receives an Abort Output, it should flush all data buffered for transmission to the front end. It should insert a Data Mark into the data stream and send a SYNCH control function to the front end.

By the service

When the service receives a Send Control Function request (other than a SYNCH control function), it should insert it into the data stream at the earliest possible point in accordance with the Telnet protocol. When the service module receives a SYNCH control function, it should flush all data it has buffered from the front end and to the net (except Telnet control functions) until a Data Mark is detected in the data stream. The service module should cause an ARPANET Host-Host INS message to be sent to the foreign host, and also send a Telnet Data Mark to the foreign host. (The service module may either pass on the Data Mark sent from the host or may insert one in the data stream, while continuing to discard data coming from the host until the Data Mark is found. If the service module inserts a Data Mark in the data stream, the Data Mark received from the host must not be sent to the foreign host.)

III J 4 c (3). Parameter field syntax

Control Function: FIXED(8)

III J 4 c (4). Parameter field semantics

This field contains a Telnet control function. The possible values and their meanings are:

242	SYNCH
243	Break
244	Interrupt Process
245	Abort Output
246	Are You There

Although there are other Telnet control functions, they are dependent on their position in the data stream and thus would not have any meaning in this context.

III J 4 d. Modify suggested allocation

This request is sent by the process to the service module to modify the allocation suggested in the BEGIN Command. The service module may use this information in any way it wishes to modify its flow control strategy with the network.

III J 4 d (1). Parameter field syntax

Messages:	FIXED(16)
Bits:	FIXED(32)

III J 4 d (2). Parameter field semantics

The semantics of the Messages and Bits fields are identical with those described in sections III A 4 b (4) and III A 4 b (5).

III J 4 e. Negotiate a Telnet option

This process-to-service protocol provides two ways for the process to negotiate a Telnet option. It may either do the negotiation itself by placing the option commands in the data stream or it may request the service module to perform the negotiation for it. The latter way uses this variant of the EXECUTE Command. It should be noted that this method can only be used when the effect of the option need not be synchronized with the data stream.

III J 4 e (1). Parameter field syntax

Negotiation: FIXED(8)
Option Code: FIXED(8)
Subnegotiation: VARIABLE(?)

III J 4 e (2). Parameter field semantics

III J 4 e (2) (a). Negotiation

This field allows the process to control option negotiation in general or to control the negotiation of a particular option. The legal values of the field are:

1	indicates service module should refuse all options
2	indicates service module should not refuse any options (i.e., it should pass those not done by the front end to the host process)
3	indicates service module should refuse any options not handled by the front end
251	WILL
252	WON'T
253	DO
254	DON'T

The meanings of WILL, WON'T, DO and DON'T are specified in the Telnet Protocol Specification [ARPANET NIC 18639]. The initial state (i.e., whether options are accepted or not) is an installation parameter.

III J 4 e (2) (b). Option code

This field contains the code for the Telnet option to be negotiated. The values of these codes may be found in the specifications of the respective options.

III J 4 e (2) (c). Subnegotiation

This field, if present, will contain one or more subnegotiations for the option specified by the option code field. These subnegotiations are to be performed for the process. Subnegotiation parameters will be bracketed within this field by the Telnet control functions IAC SB and IAC SE, where

IAC = 255
SB = 250
SE = 240

III J 4 f. Return status

No additional parameters are needed for this request.

III K. EXECUTE Response

III K 1. When sent

The service module sends an EXECUTE Response when

a) it has completed the operation requested by an EXECUTE Command or
b) it has received an EXECUTE Command whose TEXT is in error.

Note: In the case of options negotiated by the use of the EXECUTE Command, the EXECUTE Response is sent after all subnegotiations have been completed or the option has been refused.

The process sends an EXECUTE Response to acknowledge the receipt of a control function.

III K 2. Action on receipt

When the process receives an EXECUTE Response, it should examine the TEXT to see whether the Response indicates an error. If no error is indicated, it may consider that the action requested by the EXECUTE Command was successful. If an error is indicated, appropriate error recovery action should be initiated. This action is idiosyncratic to the process.

III K 3. TEXT field syntax

Result:	FIXED(3)
Ref-code:	FIXED(3)
Parameters:	VARIABLE(?)

III K 4. TEXT field semantics

III K 4 a. Result

This field contains an encoding of the result of the request (made in a previous EXECUTE Command) to which the EXECUTE Response is a reply. The codes are:

0	Success
1	Illegal code in request
2	TEXT too short
3	Illegal or unsupported parameter value

III K 4 b. Ref-code

This field contains the same value as the Code field of the TEXT of the EXECUTE Command to which the EXECUTE Response is a reply.

III K 4 c. Parameters

This field, if present, contains additional information about the outcome of the request. A discussion of these parameters follows. No parameters are needed for a Response that acknowledges receipt of a control function.

III K 4 d. New allocation

These parameters are sent by the service module to the process in response to a Modify Suggested Allocation request.

III K 4 d (1). Parameter field syntax

Messages:	FIXED(16)
Bits:	FIXED(32)

III K 4 d (2). Parameter field semantics

The semantics of the Messages and Bits fields are identical to those described in Section III B 4 b (4).

III K 4 e. Option negotiation

This reply is sent by the service module in response to the Negotiate Telnet Option request.

III K 4 e (1). Parameter field syntax

Negotiation:	FIXED(8)
Option Code:	FIXED(8)
Subnegotiation:	VARIABLE(?)

III K 4 e (2). Parameter field semantics

III K 4 e (2) (a). Negotiation

This field will contain a code indicating the final outcome of the negotiation. Legal values are:

250	indicates service module is currently accepting options
251	WILL
252	WON'T
253	DO
254	DON'T
255	indicates service module is currently refusing all options

III K 4 e (2) (b). Option code

This field contains the code of the Telnet option that was negotiated.

III K 4 e (2) (c). Subnegotiation

This field will contain the final state of any subnegotiations that were performed. The format will be consistent with the request format for subnegotiation.

III K 4 f. Status

This reply is sent by the service module in response to a status request.

III K 4 f (1). Parameter field syntax

The parameter field for this EXECUTE Response is formatted identically to the connection-info field specified in the TEXT of the BEGIN Response.

III K 4 f (2). Parameter field semantics

This field has the same meaning as the connection-info field in the TEXT of the BEGIN Response with the following exception:

III K 4 f (2) (a). Connection-state

This field indicates the present state of the connection. The possible values of this field and their meanings are:

0	Closed
1	Open
2	Waiting for open to complete
3	Waiting for close to complete

IV. Offloading Telnet Options

An installation may take one of two two basic approaches to offloading the Telnet options. One approach is to offload nothing. In this case, the Telnet service in the front end need only reformat the data stream, handle the network connections and (perhaps) do character translation. The other approach is to offload as much as possible, given the characteristics of the host. In this case, the front end must also be able to negotiate and execute some of the Telnet options. Unfortunately, for some of the Telnet options it is not possible to say definitively whether they can be offloaded or not. Table 2 can be used as a guide to how readily the various options can be offloaded.

Two attributes of each option determine whether or not it can be offloaded. An option may or may not affect the host's terminal-handling parameters. The taking effect of an option may or may not have to be synchronized with the data stream. If an option affects the host's terminal-handling parameters, then, although some of its processing may be offloaded, some processing will have to be done in the host. If an option requires synchronization with the data stream, it may be very difficult, if not impossible, to offload it. Table 2 lists the current ARPANET Telnet options and indicates whether they may be offloaded. Some of the options are discussed below in greater detail. Note that all options that are not supported by the host may be offloaded, i.e. the front end can refuse negotiations without involving the host.

Option Name	Telnet Option Num.	Host Dep.?	Synch w/Data Stream	Can Off-load?
Approximate Message Size	4	no	no	yes
Binary Transmission	0	yes	yes	no
Byte Macro	19	yes	yes	yes
Data Entry Terminal	20	yes	yes	maybe
Echo	1	no	yes	no
Extended ASCII	17	no	yes	yes
Logout	18	yes	no	no
Output Carriage Ret. Dispos.	10	no	no	yes
Output Formfeed Dispos.	13	no	no	yes
Output Horz. Tab Stops	11	yes	no	no
Output Horz. Tab Stops Dispos.	12	no	no	yes
Output Linefeed Dispos.	16	no	no	yes
Output Line Width	8	?	no	maybe
Output Page Size	9	?	no	maybe
Output Vert. Tabstops	14	yes	no	no
Output Vert. Tabstops Dispos.	15	no	no	yes
Reconnection	2	no	yes	yes
RCTE	7	yes	yes	no
Status	5	yes	yes	some
Suppress Go Ahead	3	no	no	yes
Timing Mark	6	yes	yes	no

Table 2. Classification of Telnet Options

Binary Transmission. In most cases, this option cannot be offloaded. However, the fact that it has been negotiated may be of interest to the service module in the front end. If the front end has been doing (or normally does) some translation tasks for the host, it will be necessary to coordinate the beginning of binary data with the cessation of such translation.

Byte Macro. If this option is supported and any other options or functions are offloaded, then this option must be offloaded. This option provides a means for mapping arbitrary strings into one byte. Therefore, when this option is in effect,

the macro expansion must take place at the earliest point.

Data Entry Terminal (DET). It may be possible to offload all or part of this option depending on the nature of the host system. If the host supports only one form of data entry terminal, then the front end can map many DET option subcommands into the form required by the local terminals or programs.

Echo. In general, it is not possible to offload this option, since the process in the host which is connected to this terminal may wish to echo characters other than those typed. However, there are cases where it might be desirable if the front end did handle this option for certain kinds of hosts. For example, consider computers that support only half-duplex terminals, such as IBM 360's, Burroughs 6700's, or Honeywell 6000's. Suppose that a user connects to such a host and asks it to DO ECHO. It would be quite reasonable to have the front end handle the echoing and present the host with an apparent line-at-a-time environment. This would avoid many of the difficulties of trying to fit a line-at-a-time system into a character-at-a-time mode.

Output Line Width. Once again, the decision to offload this option rests heavily on how line-width regulation is enforced. If the regulation of line width is done by simple "line folding", then this function may be performed by the front end. However, if regulating line width requires more fundamental adjustments or more sophisticated action, it is quite likely that it will not be

possible to offload this option.

Output Page Size. The decisions that are relevant for offloading this option are similar to the ones described for output-line width. Some systems may interpret the regulation of page size as follows. The maximum number of lines in a page is sent. If there is still data to be sent, the sender pauses, waiting for an input from the user indicating that he is ready to proceed. If this is the sort of regulation desired when this option is negotiated, then it can be offloaded. However, if more fundamental adjustments or more sophisticated action is required, it may not be possible to offload it.

Remote Controlled Transmission and Echoing. The RCTE option is meant as a more general mechanism to replace the Echo option. If RCTE is used to replace the Echo option, then it may be offloaded under the same conditions as the Echo option. Otherwise, offloading is not possible and actually defeats the purpose of the RCTE option.

Status. Since not all options may be offloaded, it is necessary to offload part of the Status option and to leave part of it in the front end. It is suggested that the Status option be handled in the following way:

When the service module in the front end receives a Status option negotiation, it will respond favorably to the request by sending a WILL STATUS to the sender. It will then relay the option to the host in the normal manner and wait for a response

from the host. When the host receives the DO STATUS command from the front end, it will transmit back to the front end the subnegotiation specifying the state of the options it handles. When this subnegotiation is received by the front end, it will insert into this subnegotiation the status of the options it handles and then it will send the completed subnegotiation to the foreign host.

If no options are supported in the host or if no options are supported in the front end, then, of course, this procedure may be avoided and the whole operation done in the front end or in the host, respectively. Also it is possible for the front end to know what options are not supported and supply the default status information.

Suppress Go Ahead In a sense the Suppress Go Ahead Option is always offloadable. If the host always generates go-aheads then it is no problem for the front end to filter them out when the option is negotiated. However, it is impossible for the front end to *insert* go aheads when the option has its default value.

Telnet "Synch" Sequences. In some cases, searching for "interesting" characters in the data stream may be offloaded. For example, if the only characters deemed interesting are the Telnet special characters, then this function may be offloaded. Or the front end may be provided with a list (on either a static or dynamic basis) of strings to be searched for. However, even

if these functions are performed in the front end, they must also be done in the host before the "synch" sequence is received.) The only reason for offloading this function is to decrease the response time to the user's request.

V. Scenario

Let us consider how this protocol might be used by a process in the host to initiate a Telnet connection. We will assume that as many Telnet functions as possible have been offloaded.

The user initiates a Telnet connection by causing a BEGIN Command to be sent to the front end:

<BEGIN><C><control-bits=init><host=12>

Since the default settings for a BEGIN Command favor listening for a Telnet connection, the Command must in this case explicitly request that the connection be initiated. The NVT Service Module (NSM) in the front end will open the Telnet connection to host 12. Once the connection is established, the NSM will send a Begin Response to the host:

<BEGIN><R><connection state=0><host=12><foreign skt=4283>
<msg=10><bits=10000><lskt=5607>

The channel is now established and data can flow. Almost immediately the system herald message arrives from host 12 and is passed to the local host:

<TRANSMIT><C><text, Go-ahead>Center for Adv.
Computation<CRLF>Please log in:

(Since TRANSMIT Responses require no action by either the service or the process, they will not be shown in this example.) The process wishes to have the remote Telnet implementation cease

echoing characters. So it requests the front end to negotiate with the remote Telnet process:

<EXECUTE><C><code=2><DON'T><option code=1>

The NSM in the front end enters into an option negotiation with the remote Telnet process and is successful in stopping the remote character echoing. The NSM then informs the process:

<EXECUTE><R><result=0><ref-code=2><WON'T>
<option code=1>

The user now continues with his Telnet session, sending data to the remote host and receiving responses from it. Data and responses are transferred between the host and the front end by TRANSMIT Commands.

VI. References

1. Postel, Jon, "Telnet Protocol Specification," NIC #15372, Aug. 1973.

2. Postel, Jon, "Telnet Option Specifications," NIC #15373, Aug. 1973.

3. Postel, Jon, "Telnet Binary Transmission Option," NIC #15389.

4. Postel, Jon, "Telnet Echo Option," NIC #15390.

5. Postel, Jon, "Telnet Reconnection Option," NIC #15391.

6. Postel, Jon, "Telnet Suppress Go Ahead Option," NIC #15392.

7. Postel, Jon, "Telnet Status Option," NIC #16237.

8. Postel, Jon, "Telnet Timing Mark Option," NIC #16238.

9. Postel, Jon, "Telnet Extended-Options-List Option," NIC #16239.

10. Walden, David C., "Telnet Output Line Width Option," NIC #20196, November 1973.

11. Walden, David C., "Telnet Output Page Size Option," NIC #20197, November 1973.

12. Crocker, D.H. and Postel, J.B., "Remote Controlled Transmission and Echo Telnet Option," NIC #19859, November 1973.

13. Crocker, D.H., "Telnet Output Carriage-Return Disposition Option," NIC #31155 (RFC 652), October 1974.

14. Crocker, D.H., "Telnet Output Horizontal Tabstops Option," NIC #31156 (RFC 653), October 1974.

15. Crocker, D.H., "Telnet Output Horizontal Tab Disposition Option," NIC #31157 (RFC 654), October 1974.

16. Crocker, D.H., Telnet Output Formfeed Disposition Option," NIC #31158 (RFC 655), October 1974.

17. Crocker, D.H., Telnet Output Vertical Tabstops Option," NIC #31159 (RFC 656), October 1974.

18. Crocker, D.H., Telnet Output Vertical Tab Disposition Option," NIC #31160 (RFC 657), October 1974.

19. Crocker, D.H., "Telnet Output Linefeed Disposition", NIC #31161 (RFC 658), October 1974.

20. Crocker, D.H., "Telnet Byte Macro Option", RFC 729, 1977.

21. Day, John, "Offloading ARPANET Protocols to a Front End", CAC Document No. 230, Center for Advanced Computation, University of Illinois at Urbana-Champaign, 1977.

22. Day, John, "Telnet Data Entry Terminal Option", RFC 731, 1977.

23. Grossman, G.R., "ARPANET Host-Host Process-to-Service Protocol Specification", CAC Technical Memorandum No. 80, Center for Advanced Computation, University of Illinois at Urbana-Champaign, 1977.

24. Grossman, G.R., "Host-to-Front-End Protocol Specification", RFC 710, 1977.

Appendix 5

User FTP
Process-to-Service
Protocol Specification

CAC Technical Memorandum 101

by

John Day

User FTP

Process-to-Service Protocol

Specification

I. Service Code Number

7

II. Description of the Service

This document is a process-to-service protocol (PSP) specification as defined in the Host-to-Front-End Protocol (HFP) Specification [ARPANET RFC 710]. It specifies a process-to-service protocol for providing user file transfer protocol (FTP) services to a process through the HFP.

The file transfer protocol [5] is a protocol for transferring files between hosts on the ARPANET. FTP provides the mechanisms not only for file transfer, but also for various file management functions such as deleting and renaming files. A user FTP session requires two facilities: 1) an FTP interpreter to send FTP commands, receive FTP replies and control the transfer; and 2) a data transfer facility at each host to actually move the data and to perform any data translation required. This document specifies a process-to-service protocol that provides a process-oriented, rather than a human-oriented, interface to a user FTP module (FTP interpreter) in the front end. This process-to-service protocol would be used by resource sharing or distributed computing application programs. In essence, this PSP allows the user FTP interpreter to be offloaded

while leaving the user interface in the host. This process-to-service protocol is analogous in many ways to the network virtual terminal process-to-service protocol [3].

This PSP is used in conjunction with the File Access PSP to offload the user side of FTP. This PSP is used in two offloading schemes (see CAC Document No. 230) that differ only in whether data translation is done in the host or in the front end. In either case it may be necessary for information to be passed either between the user FTP service module and the file access service module in the front end or between the user FTP interface and the file access module in the host.

The user FTP process-to-service protocol uses HFP Messages to carry information between the process and the service module. A brief summary of the function of specific Message types follows.

<u>BEGIN Command</u>
from the host process
requests that an FTP connection be established.

<u>BEGIN Response</u>
from the service module
a) confirms the establishment of the connection or
b) reports the reason for failure.

<u>END Command</u>
from the process
requests the closing of the connection.
from the service module
reports the closing of the connection by the foreign host, or reports a system failure.

<u>END Response</u>
from the process
acknowledges the closing of the connection.
from the service module

acknowledges the closing of the connection.

TRANSMIT Command

from the process
carries FTP commands (perhaps encoded) to the service module for transmission to the foreign host over the ARPANET.

from the service module
carries FTP replies (perhaps encoded) received over the ARPANET to the process.

TRANSMIT Response

is used to acknowledge receipt of FTP commands and replies.

EXECUTE Command

from the host process
requests that the service module
a) report the status of a connection;
b) send a special control function to the foreign host; or
c) modify the allocation on the connection.

EXECUTE Response

from the service module
a) acknowledges the success or failure of the action requested by the process via the EXECUTE Command and,
b) if requested, reports on the status of the connection.

This service requires implementations in the front end of the following additional protocols:

ARPANET Host-Host Protocol

ARPANET Initial Connection Protocol

ARPANET Telnet Protocol

ARPANET File Transfer Protocol (user side)

File Access Process-to-Service Protocol

The implementation of the file access process-to-service protocol and an interface to this PSP are required in the host.

The detailed specification of the use of HFP Messages follows.

III. Message Use

III A. BEGIN Command

III A 1. When sent

A BEGIN Command is sent by the process to request that the service module attempt to establish an ARPANET FTP connection.

III A 2. Action on receipt

When the service module receives the BEGIN Command, it will attempt to establish the FTP connection according to the parameters in TEXT.

III A 3. TEXT field syntax

```
Security:                   VARIABLE(?)
Establishment:              COMPLEX(8)
   Control-bits:               FIXED(4)
      Socket/Channel
      ICP/Direct
      Relative/Absolute
      Unattachable/Attachable
   Host:                       FIXED(32)
   Foreign Socket:             FIXED(32)
   Messages:                   FIXED(16)
   Bits:                       FIXED(32)
   Local Socket:               FIXED(32)
   Timeout:                    FIXED(16)
```

III A 4. TEXT field semantics

III A 4 a. Security

This field can be used to validate the process' access privileges to network resources. It is particularly important to limit access to network security objects. The substructure and content of this field are installation dependent.

NOTE: At this writing, not enough is known about security in networks and front ends to make any statement about the utility of validation at this point in the system. This field is included in case some systems require validation at this point.

III A 4 b. Establishment

This set of parameters governs the establishment of the network connection. The default values for the establishment parameters are chosen to favor the user FTP service. This means that a BEGIN Command sent to the front end specifying only defaults except in the foreign host field would cause the front end to initiate an ICP to the FTP contact socket (socket 3) and set up an FTP control connection in accordance with the protocol. The default values of the fields are set at zero; a default value may be indicated by the absence of the field. Thus, most BEGIN Commands sent to the front end to establish an FTP session will only specify the foreign host.

III A 4 b (1). Control-bits

The bits in this field govern the kind of connection to be established, the method of its establishment, and the interpretation of the parameters.

III A 4 b (1) (a). Socket/Channel

If this bit is zero, the logical channel from the host process is to be established to a new ARPANET connection. If this bit is one, the logical channel from the relay process is to be attached to an already existing HFP channel (e.g. a channel already established to the host-host service module and therefore providing access to an existing network connection.) The default value is zero.

III A 4 b (1) (b). ICP/Direct

When this bit is zero (default), the connection is to be established via the Initial Connection Protocol. When this bit is one, the connection is to be established directly to the specified foreign socket. The use of this bit in specifying connection sockets is detailed in table 1.

III A 4 b (1) (c). Relative/Absolute

This field affects the values of the local sockets for the connection. See III A 4 b (6) for details. The default case is Relative (value zero).

III A 4 b (1) (d). Unattachable/Attachable

When this bit is one, the service module is requested to make the logical channel available for attachment by other service modules within the front end (such as a data transformation service). If this bit is zero (default), the service module is not requested to make the logical channel available for attachment within the front end.

III A 4 b (2). Host

This field specifies the network host to which the connection is to be made. No default value of this field is specified in the PSP. If this field is to have a default value, the value and its meaning must be specified in the adaptation description.

The Host field is parsed as:

Network:	FIXED(8)
Imp:	FIXED(16)
Host-at-Imp:	FIXED(8)

III A 4 b (3). Foreign Socket

This field is used as follows to construct the effective foreign socket (e.f.s.) for making the connection.

Case 1. Foreign Socket field = zero. The e.f.s. is three.
Case 2. Foreign Socket field ≠ zero. The e.f.s. is the contents of the Foreign Socket field.

The default value of this field is zero. The e.f.s., in conjunction with the ICP/Direct bit, is used to determine the foreign sockets for the connection. See table 1 for details.

III A 4 b (4). Messages

This field and the Bits field allow the process to indicate to the service module the flow rate desirable on the FTP data connection. The service module may use this information in any way its implementors deem desirable.

This field contains the number of ARPANET Host-Host messages the process suggests be allocated on the connection. If the value of this field is zero (default), the service module chooses the number of

messages on its own.

III A 4 b (5). Bits

This field contains the number of bits the process suggests be allocated for the FTP data connection. If the value of this field is zero (default), the service module chooses the number of bits on its own.

III A 4 b (6). Local Socket

This field is used as follows to construct the <u>effective local socket</u> (e.l.s.) for making the connection.

Case 1. Relative/Absolute bit = zero, Local Socket field ≠ zero. In this case, the contents of the Group field in the HEADER will be concatenated with the low-order 20 bits of the Local Socket field to form the e.l.s.

Case 2. Relative/Absolute bit = zero, Local Socket field = zero. In this case, the service will choose a number whose high order 12 bits are equal to the Group field in the HEADER.

Case 3. Relative/Absolute bit = one. In this case the effective local socket is equal to the contents of the Local Socket field.

The relationship between the e.l.s. and the local socket values depends upon the value of the ICP/Direct bit. See table 1 for details.

III A 4 b (7). Timeout

This field contains the maximum length of time, in seconds, that the service module is to attempt to establish the connection (in the absence of an error or exceptional condition). If the service module has been unable to establish the connection within this time period, it is to abandon the attempt. If this field contains zero (default), the time period is to be infinite.

Table 1

Control Bits	Foreign Sockets for the connection	Local Sockets for the connection
ICP	The e.f.s. will be used as the contact socket for the connection. The control connection sockets will be chosen by the foreign host.	e.l.s. +2 and e.l.s. +3
Direct	e.f.s. and e.f.s. +1	e.l.s. and e.l.s. + 1

III B. BEGIN Response

III B 1. When sent

A BEGIN Response is sent by the service module to report the outcome of an attempt to establish an FTP connection requested by a BEGIN Command. The Status field in the HEADER will indicate whether the connection attempt was successful or unsuccessful.

III B 2. Action on receipt

When the process receives a BEGIN Response indicating a successful connection, it may proceed to send and receive data on the connection. When the process receives a BEGIN Response indicating an unsuccessful connection, it should perform the appropriate error recovery.

III B 3. TEXT field syntax

```
Connection-info:            COMPLEX(6)
   Connection-state:           FIXED(4)
   Host:                       FIXED(32)
   Foreign Socket:             FIXED(32)
   Messages:                   FIXED(16)
   Bits:                       FIXED(32)
   Local Socket:               FIXED(32)
```

III B 4. TEXT field semantics

III B 4 a. Connection-state

If the connection attempt was successful (as indicated by the Status field in the HEADER), this field will contain the value 1 which will be compatible with the "open" state value defined for this field in the TEXT of the EXECUTE Response. (See section III K 4 c (1).)

If the connection attempt was unsuccessful, this field contains an encoded reason for the failure. The codes are:

0	Illegal control bit combination
1	Invalid local socket
2	Invalid host
3	Connection attempt timeout
4	Network not available
5	Foreign host dead
6	Connection refused
7	Front end terminating service
8	Network connection incompletely specified

III B 4 b. Host

If the connection attempt was successful, this field contains the ARPANET host address of the host to which the connection was established. (See III A 4 b (2).)

III B 4 c. Foreign Socket

If the connection attempt was successful, this field contains the socket number at the foreign host to which the connection was established.

III B 4 d. Messages and Bits

If the connection attempt was successful, these fields contain the flow control parameters for the connection at the time the BEGIN Response was generated.

III B 4 e. Local Socket

If the connection attempt was successful, this field contains the local socket number for the connection.

III C. END Command

III C 1. When sent

III C 1 a. By the process

An END Command is sent by the process to request the service module to close the network connection.

III C 1 b. By the service module

An END Command is sent by the service module to notify the process that the connection has been closed by the foreign host or by a failure of the foreign host or the network.

III C 2. Action on receipt

III C 2 a. By the process

When the process receives an END Command, it should consider the connection closed. The process should send an END Response and take appropriate recovery action.

III C 2 b. By the service module

When the service module receives an END Command, it should allow data queued for transmission to the network to drain, unless Flush away (bit 1) is set in the Control field. It should then initiate the process of closing the connection. When the connection is closed, the service module should send an END Response.

III C 3. TEXT field syntax

Cause: FIXED(3)

III C 4. TEXT field semantics

III C 4 a. Cause

This field contains an encoded reason for sending the END Command. The codes are:

0	Normal completion (default)
1	Network/process failure
2	Foreign host failure
3	User requested termination
4	Front end terminating service

III D. END Response

III D 1. When sent

III D 1 a. By the process

The END Response is sent by the process to acknowledge the receipt of an END Command and complete the termination of the associated logical channel.

III D 1 b. By the service module

The END Response is sent by the service module to acknowledge receipt of an END Command, notify the process that the connection has been closed as requested, and terminate the logical channel.

III D 2. Action on receipt

III D 2 a. By the process

When the process receives the END Response, it should consider the ARPANET connection to be closed and the HFP logical channel to be terminated.

III D 2 b. By the service module

When the service module receives the END Response, it should consider the logical channel to be terminated.

III D 3. TEXT field syntax

The TEXT field in the END Response is not used in this protocol.

III E. TRANSMIT Command

III E 1. When sent

III E 1 a. By the process

The TRANSMIT Command is sent by the process to pass FTP commands to the service module for transmission over the network to the foreign host.

III E 1 b. By the service module

The TRANSMIT Command is sent by the service module to pass FTP replies that have arrived over the network from the foreign host to the process.

III E 2. Action on receipt

III E 2 a. By the process

When the process receives a TRANSMIT Command, it will treat the contents of the TEXT as FTP replies to be processed. (See section IV for details.)

III E 2 b. By the service module

When the service module receives a TRANSMIT Command, it will treat the contents of the TEXT as FTP commands to be transmitted over the network. (See section IV for details.)

III E 3. TEXT field syntax

```
Control:            FIXED(1)
   Go-Ahead/More-Data
Data:               VARIABLE(?)
```

III E 4. TEXT field semantics

III E 4 a. Control

The single bit in this field indicates whether or not the sender of the TRANSMIT Command has completed sending data. It is used to facilitate handling half-duplex terminals connected to the process. The receiver of a TRANSMIT Command with this bit set to 0 (default) should assume that the sender has completed sending data and that the receiver is now free to send any data that it has. If the bit is set to 1 the sender has more data to send and the receiver should not send data at this time.

III E 4 b. Data

The Data field contains the FTP commands and replies to be transferred. For more details see CAC Document 230, and section IV.

III F. TRANSMIT Response

The TRANSMIT Response is used as described in the HFP specification.

III G. SIGNAL Command

The SIGNAL Command is not used in the process-to-service protocol. SIGNAL Commands may be generated by process or service to request that the CPM carry out the appropriate action on the logical channel. (See HFP specification.) Thus, the effect of the SIGNAL Command is restricted to the logical channel level.

III H. SIGNAL Response

The SIGNAL Response is not used by either process or service.

III J. EXECUTE Command

III J 1. When Sent

The process sends an EXECUTE Command to request the service module to

a) report the status of a specified connection,
b) send a Telnet control function,
c) handle Telnet options and FTP replies, or
d) modify the allocation for the connection.

The service module does not send EXECUTE Commands for this protocol.

III J 2. Action upon Receipt

When the service module receives an EXECUTE Command it will decode and act on the request.

III J 3. TEXT field syntax

Code:	FIXED(2)
Parameters:	VARIABLE(?)

III J 4. TEXT field semantics

III J 4 a. Code

This field contains an encoding of the type of request. The codes are:

0	Report status
1	Send control function
2	Handle Telnet options and FTP replies
3	Modify suggested allocation

III J 4 b. Parameters

This field contains the parameters for specifying the requests. The contents will vary according to the type of request.

III J 4 b (1). Report status

No parameters are required.

III J 4 b (2). Send control function

III J 4 b (2) (a). Parameter field syntax

Control Function: FIXED (8)

III J 4 b (2) (b). Parameter field semantics

This field contains a parameter specifying a Telnet control function that the User FTP module should send on the control connection. The parameter values and their meanings are:

242	SYNCH
243	Break
244	Interrupt process
245	Abort output
246	Are you there

III J 4 b (3). Handle options and FTP replies

III J 4 b (3) (a). Parameter field syntax

Option control:	FIXED(2)
Reply text control:	FIXED(1)

III J 4 b (3) (b). Parameter field semantics

These fields contain parameters that indicate 1) how the front end should respond to Telnet option negotiations on the control connection or 2) whether or not the front end should forward the text of FTP replies to the host. How Telnet options and FTP replies are handled before the process sends the relevant EXECUTE Commands is installation-specific and must be specified in the adaptation description. The parameter values and their meanings are:

Option control values:

1	Refuse all options
2	Refuse no options
3	Refuse options not handled by the front end

Reply text control values:

0	Send FTP reply text
1	Don't send FTP reply text

III J 4 b (4). Modify suggested allocation

III J 4 b (4) (a). Parameter field syntax

Messages:	FIXED(16)
Bits:	FIXED(32)

III J 4 b (4) (b). Parameter field semantics

The semantics of this field are identical to

those described for the allocation fields in the BEGIN Command. (See sections III A 4 b (4) - III A 4 b (5).)

III K. EXECUTE Response

III K 1. When Sent

The service module sends an EXECUTE Response when

a) it has completed the operation requested by an EXECUTE Command, or
b) it has received an EXECUTE Command whose TEXT is in error.

The process does not send EXECUTE Responses for this process-to-service protocol.

III K 2. Action on Receipt

When the process receives an EXECUTE Response, it should examine the TEXT to see whether the Response indicates an error. If no error is indicated, it may consider that the action requested by a previous EXECUTE Command was successful. If an error is indicated, appropriate error recovery action should be initiated. This action is idiosyncratic to the process.

III K 3. TEXT field syntax

Result:	FIXED(2)
Ref-Code:	FIXED(2)
Connection-info:	COMPLEX(?)
Control connection-state:	FIXED(2)
Data connection-state:	FIXED(2)
Host:	FIXED(32)
Foreign Socket:	FIXED(32)
Messages:	FIXED(16)
Bits:	FIXED(32)
Local Socket:	FIXED(32)

III K 4. TEXT field semantics

III K 4 a. Result

This field contains an encoding of the result of the request (made in a previous EXECUTE Command) to which the EXECUTE Response is a reply. The codes are:

0	Success
1	Illegal code in request
2	TEXT too short.

III K 4 b. Ref-code

This field contains the same value as the Code field of the TEXT of the EXECUTE Command to which the

EXECUTE Response is a reply.

III K 4 c. Connection-info

This field is present only if the Ref-code has a value of zero or three (status or modify allocation). The semantics of this field are identical with the Connection-info field of the BEGIN Response (section III B 4) with the following exception.

III K 4 c (1). Control connection-state

If the ref-code value is zero (status), this field indicates the present state of the FTP control connection. The possible values of this field and their meanings are:

0	Closed
1	Open
2	Waiting for open to complete
3	Waiting for close to complete

III K 4 c (2). Data connection-state

If the ref-code value is zero (status), this field indicates the present state of the FTP Data connection. The possible values of this field and their meanings are:

0	Closed
1	Open
2	Waiting for open to complete
3	Waiting for close to complete

IV. Details of Offloading User FTP

The ARPANET File Transfer Protocol consists of two major sections: a command-and-response section and a data transfer section. The file access process-to-service protocol deals with offloading the data transfer portion, while this protocol and the program access process-to-service protocol deal with different ways of offloading the command-and-response section.

The program access process-to-service protocol offloads the entire command-and-response section including the user interface. Thus programs which wish to use FTP are required to deal with the human-oriented interface of the front end. The user FTP process-to-service protocol defines an offloading scheme more appropriate for the use of FTP by programs. Alternatively, one can view this protocol as defining an offloading scheme that leaves the user interface in the host.

In this protocol, the TRANSMIT Command can be used to send FTP commands and replies between the service module and the process. Some installations may wish to send some intermediate form or encoding peculiar to their host's file system or user interface and let the service module in the front end translate into the proper FTP Commands. For most installations, the best method of encoding this information is to just send the FTP commands themselves. Since many of the parameters are character strings, little is saved by additional encoding. The only major exceptions would be the data transfer parameter commands such as

TYPE, MODE, etc. Similar arguments hold for the replies. Because of the semantics attached to the individual decimal digits, some coding would be possible, but of limited utility. Some installations may wish to use the control function described in Section III J 4 b (3) to send only the reply code to the process and not the full text of the reply. Section V gives a suggested structure for encoded FTP commands and replies.

File transfers can be handled in two ways: either the host or the front end controls the transfer. In either case the transfer is mediated by the file access process-to-service protocol. If the transfers are controlled from the host (i.e., the using file access module in the host), then the process will supply the file access service module in the host with the local file name. Thus, the FTP file transfer commands (e.g. RETR, SEND, etc.) can be sent as they are defined in the protocol to the front end. However, if the transfers are controlled from the front end (i.e., by the using file access module in the front end), the process must inform the service module in the front end of the names of both the local file and the remote file to be used in the transfer. This will require that both pathnames (local and foreign) be passed to the front end. For further discussion of FTP offloading see [4].

V. Suggested Encoding of FTP Commands and Replies

V A. Command Syntax

Code: FIXED(5)
Parameters: VARIABLE(?)

V A 1. Command Semantics

V A 1 a. Code

This field contains a code which represents a particular FTP command. The values are:

0	USER
1	PASS
2	ACCT
3	REIN
4	BYE
5	BYTE
6	SOCK
7	PASV
8	TYPE
9	STRU
10	MODE
11	RETR
12	STOR
13	APPE
14	ALLO
15	REST
16	RNFR
17	RNTO
18	ABOR
19	DELE
20	LIST
21	NLST
22	SITE
23	STAT
24	HELP

V A 1 b. Parameters

This field contains any parameters necessary for the commands. They are encoded as follows:

V A 1 b (1). Parameters for USER, PASS, ACCT

Access Parameter: VARIABLE( )

V A 1 b (2). Parameters for Data Transfer Commands

Byte Size: FIXED(8)
Socket: FIXED(32)

Type:	FIXED(4)
Stru:	FIXED(1)
Mode:	FIXED(2)
Allo:	FIXED( )
File Transfer:	COMPLEX(2)
Foreign path:	VARIABLE(?)
Local path:	VARIABLE(?)

V B. Reply syntax

Reply code:	FIXED(10)
Reply text:	VARIABLE(?)

The three-digit reply codes are coded as

Bits 0-2:	1st digit
Bits 3-5:	2nd digit
Bits 6-9:	3rd digit

VI. Scenario

Let us consider how this protocol might be used to offload a user FTP which has its user interface in the host and uses the file access process-to-service protocol to transfer files. The using file access module (UFAM) and the data translation functions are in the front end.

The user initiates the host's user FTP process (UFP) and asks for a connection to ILL-CAC. The UFP notes that this is a normal FTP request and sends a BEGIN Command to the front end specifying only the foreign host (the other parameters have their default values):

<BEGIN><C><security=day,john><control-bits=0><host=12>

The user FTP service module (UFSM) in the front end will then attempt an ICP to socket 3 (the FTP contact socket) at ILL-CAC. Once the connection is established the UFSM will return a BEGIN Response to the UFP indicating the outcome:

<BEGIN><R><state=1><host=12><foreign skt=4399><msg=4>
<bits=2000><1skt=2363>

Later the herald message arrives from the FTP server at ILL-CAC and is passed to the UFP:

<TRANSMIT><C>300 Center for Adv. Computation. Please log in:

(Since TRANSMIT Responses require no action by either the service or the process, they will not be shown in this example.) The user now sends the FTP USER command which is passed from the UFP to the UFSM and on to the net:

<TRANSMIT><C>USER day

(The UFSM notes what command is being sent so that it can know what replies it should expect in return. For most commands this is mainly an integrity check on the operation of the server. However, for the data transfer commands the replies indicate the state of the transfer and may have an effect on the control of the data connection.) The response comes from the server FTP and is passed on to the UFP:

<TRANSMIT><C>331 Please enter password

The user enters the FTP PASS command and it is passed to the UFSM:

<TRANSMIT><C>PASS john

This command is then sent to the server FTP. When the response arrives it is passed to the UFP:

<TRANSMIT><C>230 User logged in for file transfer

The user now wishes to move a file from the server to a local file on the host. Since such transfers are initiated and controlled by the UFAM in the front end, both the local and remote pathnames must be passed to the front end. The following

command is sent to the UFSM:

<TRANSMIT><C>RETR junk.note to note.junk

where the first pathname is the remote one and the second the local one. The UFSM will send the portion:

RETR junk.note

to the server FTP, initiate a data connection on the default data socket, and pass the second pathname, the local port and security information to the UFAM. The UFAM will then establish a session with the host and begin transfer of data into the host. Once the transfer has begun the server FTP will send a 125 reply which will be passed on to the UFP:

<TRANSMIT><C>125 File Transfer begun

After the transfer has completed the server FTP will send a 226 which will be passed on to the UFP:

<TRANSMIT><C>226 File Transfer completed

Suppose that the user then wishes to delete the file he has just transferred. The UFP sends an FTP DELE command to the UFSM:

<TRANSMIT><C>DELE junk.note

The UFSM passes the TEXT of this message on to the server FTP. Later a reply is returned and passed on the UFP:

<TRANSMIT><C>250 File junk.note deleted

The user then wishes to terminate the FTP session. The UFP passes an FTP QUIT command to the UFSM:

<TRANSMIT><C>QUIT

The UFSM passes the QUIT on to the server FTP. The server FTP responds with a 221 reply. However, before this reply is received the UFP terminates the channel by sending an END Command:

<END><C><status=0> (indicating normal termination)

and the UFSM replies promptly with:

<END><R>

The UFSM is then left to close all network connections in accordance with the FTP protocol and to notify the UFAM to terminate its session also.

VII. References

1. Day, J. "Server FTP Process-to-Service Protocol," CAC Technical Memorandum No. 100, Center for Advanced Computation, University of Illinois at Urbana-Champaign, 1977.

2. Day, J. "File Access Process-to-Service Protocol," CAC Technical Memorandum No. 102, Center for Advanced Computation, University of Illinois at Urbana-Champaign, 1977.

3. Day, J. "Network Virtual Terminal Process-to-Service Protocol," CAC Technical Memorandum No. 103, Center for Advanced Computation, University of Illinois at Urbana-Champaign, 1977.

4. Day, J. "Offloading ARPANET Protocols to a Front End," CAC Document No. 230, Center for Advanced Computation, University of Illinois at Urbana-Champaign, 1977.

5. Neigus, N. "The File Transfer Protocol," ARPANET RFC 542, 1973.

6. Neigus, N; Pogran, K.; and Postel, J. "A New Schema for FTP Reply Codes," ARPANET RFC 640, 1974.

7. Postel, J. "Telnet Protocol Specification," NIC #18639, 1973.

Appendix 6

Server FTP
Process-to-Service
Protocol Specification

CAC Technical Memorandum 100

by

John Day

Server FTP

Process-to-Service Protocol

Specification

I. Service Code Number

8

II. Description of the Service

This document is a process-to-service protocol specification as defined in the Host-to-Front-End Protocol (HFP) Specification [ARPANET RFC 710]. It describes a protocol for offloading the server side of the ARPANET File Transfer Protocol [4].

The file transfer protocol (FTP) is a protocol for transferring files between hosts on the ARPANET. FTP provides the mechanisms not only for file transfer, but also for various file management functions such as deleting and renaming files. A Server FTP session requires an FTP protocol interpreter to control the transfer and to perform the file management operations and a data transfer process to actually move the data and to perform any translations between the network data representation and the local data representation. The file access process-to-service protocol deals with offloading the data transfer section. The server FTP process-to-service protocol deals with offloading the command-and-response section. Of the twenty-five FTP commands defined in the protocol, only seven must be done by the residual server FTP process in the host. The others are handled by either the file access service module or

the server FTP process in the front end. Those seven commands are:

RENAME

DELETE

SITE

STATUS <arg>

ALLOCATE

LIST

NAME-LIST

The FTP commands used for user authentication (USER, PASS, and ACCT) may or may not be offloaded depending on the security policy arrangement between the front end and the host.

The organization of some host systems may be such that the ALLOCATE, LIST and NAME-LIST commands are best performed through the file access process-to-service protocol. Depending on the particular offloading environment the other commands may be handled in either the host or the front end.

This process-to-service protocol requires implementations in the front end of the following additional protocols:

ARPANET Host-Host Protocol

ARPANET Initial Connection Protocol

ARPANET Server Telnet

ARPANET File Transfer Protocol (Server side)

File Access Process-to-Service Protocol

Implementations of the File Access PSP and a residual Server FTP which uses the Server FTP PSP will normally be required in the host. (Installations that only allow the transfer of data by network users will not require the use of this protocol.)

The server FTP process-to-service protocol uses HFP Messages to carry information between the residual part of Server FTP in the host (the "process") and the server FTP service module (the "service") in the front end. A brief summary of the specific Message types follows.

<u>BEGIN Command</u>
is used to initiate connections.

<u>BEGIN Response</u>
is used to confirm the establishment of a connection, or to report the reason for failure.

<u>END Command</u>
is used to terminate a connection.

<u>END Response</u>
is used to confirm the termination of a connection.

<u>TRANSMIT Command</u>
is used to transfer FTP commands and replies to and from the Telnet connection.

<u>TRANSMIT Response</u>
is used to acknowledge the transfer of FTP commands and replies.

<u>EXECUTE Command and Response</u>
a) report the status of connections,
b) perform control functions, and
c) modify suggested allocations.

The details of the use of HFP Messages are given in the following section.

III. Message Use

III A. BEGIN Command

III A 1. When sent

III A 1 a. By the host process

A BEGIN Command is sent by the host process to indicate that it is able to accept an FTP user. The host process sends as many BEGIN Commands as the number of connections it will accept.

III A 1 b. By the service module

In some offloading schemes, a BEGIN Command is sent by the service module to request FTP service in behalf of a particular user.

III A 2. Action on receipt

III A 2 a. By the service module

When the service module receives a BEGIN Command, it attempts to set up a connection as specified by the parameters in the TEXT. The service module may refuse the BEGIN Command by sending an appropriate BEGIN Response.

III A 2 b. By the host process

When the host receives a BEGIN Command, it should attempt to provide FTP service for those FTP commands not handled in the front end. The host may refuse the request for service by sending an appropriate BEGIN Response. The only field the host process should expect in the TEXT of the BEGIN Command is the Security field.

III A 3. TEXT field syntax

```
Security:                                 VARIABLE(?)
Establishment:                            COMPLEX(8)
   Control-bits:                             FIXED(4)
      Socket/Channel
      ICP/Direct
      Relative/Absolute
      Unattachable/Attachable
   Host:                                     FIXED(32)
   Foreign Socket:                           FIXED(32)
   Messages:                                 FIXED(16)
   Bits:                                     FIXED(32)
   Local Socket or Logical Channel:          FIXED(32)
   Timeout:                                  FIXED(16)
```

III A 4. TEXT field semantics

III A 4 a. Security

This field may be used by the front-end or the host to validate the access privileges of the server or user, respectively. The default value of the field is empty. Note: At this time, very little work has been done on the organization of security for networks or these kinds of systems. It is not clear where access control should be placed. This field has been included in case some systems require validation at this point.

III A 4 b. Establishment

This set of parameters describes the kind of connection to be associated with this channel. The default for the establishment parameters are chosen to favor the Server FTP service. This means that a BEGIN Command sent to the front end specifying only defaults would cause the front-end Server FTP Service to listen for an ICP connection on the standard FTP contact socket (socket 3) from any host for an indefinite period of time. The default values of the fields are set at zero; a default value may be indicated by the absence of the field. Thus, most BEGIN Commands sent to the front end for a Server FTP session will have an empty TEXT field.

III A 4 b (1). Control-bits

The bits in this field govern the kind of connection to be established and the interpretation of the parameters.

III A 4 b (1) (a). Socket/Channel

If this bit is zero, the logical channel from the relay process is to be established to a new ARPANET connection. If this bit is one, the logical channel from the host process is to be attached to an already existing HFP channel (e.g. a channel already established to the FTP service module and therefore providing access to an existing network connection.) The default value is zero.

III A 4 b (1) (b). ICP/Direct

This bit is relevant only when a new network connection is requested. (Socket/Channel bit is zero.) When this bit is zero (default), the connection is to be established via the Initial Connection Protocol. When this bit is one, the

connection is to be established directly. The use of this bit in specifying connection sockets is detailed in table 1.

III A 4 b (1) (c). Relative/Absolute

This field affects the value of the socket for the local connection. See III A 4 b (6) for details.

III A 4 b (1) (d). Unattachable/Attachable

When this bit is one, the server FTP service module is requested to make the logical channel available for attachment by other service modules within the front end (such as a data transformation service). Otherwise this bit is zero (default). See CAC Technical Memorandum No. 80, p. 3, for a discussion of this feature.

III A 4 b (2). Host

This field specifies the network host or hosts from which the FTP connection is to be expected. The default value is zero, indicating that a connection from any host will be accepted. The Host field is parsed as:

Network:	FIXED(8)
IMP:	FIXED(16)
Host-at-IMP:	FIXED(8)

III A 4 b (3). Foreign Socket

This field is used as follows to construct the effective foreign socket (e.f.s.) for making the connection.

Case 1. Foreign Socket field = zero. The e.f.s. is three.
Case 2. Foreign Socket field ≠ zero. The e.f.s. is the contents of the Foreign Socket field.

The default value of this field is zero. The relationship between the e.f.s. and the foreign sockets for the connection depends upon the value of the ICP/Direct bit. See table 1 for details.

III A 4 b (4). Messages

This field and the Bits field allow the process to indicate to the service module the flow rate

desirable on the FTP data connection. The service module may use this information in any way its implementors deem desirable.

This field contains the number of host-host messages the process suggests be allocated on the connection. If the value of this field is zero (default), the NCP chooses the number of messages on its own.

<u>III A 4 b (5). Bits</u>

This field contains the number of bits the process suggests be allocated for the FTP data connection. If the value of this field is zero (default), the NCP chooses the number of bits on its own.

<u>III A 4 b (6). Local Socket or Logical Channel</u>

If the Socket/Channel bit is one, this field contains the channel Group and Member values specifying the HFP channel to which connection is to be made. The format of this field is then:

```
<not used>: FIXED(4)
 Group:     FIXED(12)
 Member:    FIXED(16)
```

If the Socket/Channel bit is zero, this field is used as follows to construct the <u>effective local socket</u> (e.l.s.) for making the connection.

Case 1. Relative/Absolute bit = zero, Local Socket field ≠ zero. In this case, the contents of the Group field in the HEADER will be concatenated with the low-order 20 bits of the Local Socket field to form the e.l.s.

Case 2. Relative/Absolute = zero, Local Socket field = zero. In this case, the service will choose a number whose high order 12 bits are equal to the Group field in the HEADER.

Case 3. Relative/Absolute = one. In this case the effective local socket is equal to the contents of the Local Socket field.

The relationship between the e.l.s. and the local sockets for the connection depends upon the value of the ICP/Direct bit. See table 1 for details.

<u>III A 4 b (7). Timeout</u>

This field contains the maximum length of time, in seconds, that the service module is to attempt to establish the connection (in the absence of an error or exceptional condition). If the service module has been unable to establish the connection within this time period, it is to abandon the attempt. If this field contains zero (default), the time period is to be infinite.

Table 1

Control Bits	Foreign Sockets for the connection	Local Sockets for the connection
ICP	The e.f.s. will be used as the contact socket for the connection. The control connection sockets will be chosen by the foreign host.	e.l.s. +2 and e.l.s. +3
Direct	e.f.s. and e.f.s. +1	e.l.s. and e.l.s. + 1

III B. BEGIN Response

III B 1. When sent

III B 1 a. By the service module

The BEGIN Response is sent by the service module to indicate that a successful network connection has been established, or that the BEGIN Command has been refused.

III B 1 b. By the host process

The BEGIN Response is sent by the host to indicate that it has established FTP service for the non-offloaded FTP commands, or that the host is unable to provide such service.

III B 2. Action on receipt

III B 2 a. By the host

When the BEGIN Response is received by the host, the process should determine whether a successful connection was made. If so, it may send or receive data. If not, it may be appropriate to invoke an error recovery procedure.

III B 2 b. By the service module

The only field that the service module should expect to see in the TEXT of a BEGIN Response is the Status field. The service module should take note of the Status field value; otherwise no action is called for.

III B 3. TEXT field syntax

```
Security:                      VARIABLE(?)
Connection-info:               COMPLEX(?)
   Status:                        FIXED(4)
   Host:                          FIXED(32)
   Foreign Socket:                FIXED(32)
   Messages:                      FIXED(16)
   Bits:                          FIXED(32)
   Local Socket:                  FIXED(32)
```

III B 4. TEXT field semantics

III B 4 a. Security

This field may be used to allow the host to validate the access privileges of the user and to pass the information along to the front end. This field could be particularly important for monitoring access by

users directly connected to the front end.

III B 4 b. Connection-info

This field describes the connection associated with this channel.

III B 4 b (1). Status

This field indicates the success or failure of the BEGIN Command. The value zero indicates success while any non-zero value indicates failure. The particular value of the non-zero field gives the reason for failure. The possible values of this field and their meanings are:

0	Success
1	Illegal connection type
2	Invalid local socket
3	Unknown host
4	Improper access
5	Front end terminating service
6	Connection attempt timeout
7	Network not available
8	Foreign host dead
9	Connection refused
10	Local host overloaded

Others may be defined later.

III B 4 b (2). Host

If the connection attempt was successful, this field contains the ARPANET host address of the host to which the connection was established. (See III A 4 b (2).)

III B 4 b (3). Foreign Socket

If the connection attempt was successful, this field contains the socket number at the foreign host to which the connection was established.

III B 4 b (4). Messages and Bits

If the connection attempt was successful, these fields contain the flow control parameters for the connection at the time the BEGIN Response was generated.

III B 4 b (5). Local Socket

If the connection attempt was successful, this field contains the local socket number for the connection.

III C. END Command

III C 1. When sent

The END Command may be sent by either the process or the service to terminate an FTP connection. If sent by the process, it usually indicates that the host system has requested that the connection be closed. The service module may send an END Command because the foreign host has closed the connection in response to a user request, or because some failure condition has occurred that requires that the connection be aborted.

III C 2. Action on receipt

III C 2 a. By the process

When the process receives an END Command, it should acknowledge it as promptly as possible by sending an END Response.

III C 2 b. By the service module

When the service module receives an END Command, it should close the FTP data connection (if open), send the proper FTP replies indicating termination (120, 221, 421 or 530), and close the control connection. Once the connections are closed, the service module should send the END Response to the process.

III C 3. TEXT field syntax

Status: FIXED (3)

III C 4.TEXT field semantics

III C 4 a. Status

This field indicates the cause of the termination of the TELNET connection. The values of the field and their associated meanings are:

0	Normal close by foreign host
1	Network failure
2	Foreign host failure
3	User requested close
4	Improper access
5	Front end terminating service

Other causes for termination may be defined at a later date. The default value of the field is zero.

III D. END Response

III D 1. When sent

The END Response is sent by either the process or the service module to acknowledge that an END Command has been received and proper action taken.

III D 2. Action on receipt

When either the process or the service receives an END Response, it should assume that all dialogue associated with this logical channel has completed, and it should ignore any other Commands on this channel except a BEGIN Command.

III D 3. TEXT field syntax

The TEXT field in the END Response is not used in this protocol.

III E. TRANSMIT Command

III E 1. When sent

III E 1 a. By the process

The TRANSMIT Command is sent by the process to pass FTP replies to the service module for transmission over the network to the foreign host. (For details of offloading see CAC Document 230.)

III E 1 b. By the service module

The TRANSMIT Command is sent by the service module to pass FTP commands that have arrived over the network from the foreign host to the process. (For details of offloading see CAC Document 230.)

III E 2. Action on receipt

III E 2 a. By the process

When the process receives a TRANSMIT Command, it will treat the contents of the TEXT as an FTP command to be processed. (For details see CAC Document 230.)

III E 2 b. By the service module

When the service module receives a TRANSMIT Command, it will treat the contents of the TEXT as an FTP reply to be transmitted over the network. (For details see CAC Document 230.)

III E 3. TEXT field syntax

Control:	FIXED(1)
Go-Ahead/More-Data	
Data:	VARIABLE(?)

III E 4. TEXT field semantics

III E 4 a. Control

The single bit in this field indicates whether or not the sender of the TRANSMIT Command has completed sending data. It is used to facilitate handling half-duplex terminals connected to the process. The receiver of a TRANSMIT Command with this bit set to 0 (default) should assume that the sender has completed sending data and that the receiver is now free to send any data that it has. If the bit is set to 1 the sender has more data to send and the receiver should not send data at this

time.

III E 4 b. Data

The Data field contains the FTP commands and replies to be transferred. For more details see CAC Document 230, and section IV.

III F. TRANSMIT Response

The TRANSMIT Response is used as described in the HFP specification.

III G. SIGNAL Command

The SIGNAL Command is not used in the process-to-service protocol. SIGNAL Commands may be generated by process or service to request that the CPM carry out the appropriate action on the logical channel. (See HFP specification.) Thus, the effect of the SIGNAL Command is restricted to the logical channel (HFP) level.

III H. SIGNAL Response

The SIGNAL Response is not used by either process or service.

III J. EXECUTE Command

III J 1. When Sent

The process sends an EXECUTE Command to request the service module to

a) report the status of a specified connection,
b) send a Telnet control function,
c) handle Telnet options and FTP replies, or
d) modify the allocation for the connection.

The service module does not send EXECUTE Commands for this protocol.

III J 2. Action upon Receipt

When the service module receives an EXECUTE Command it will decode and act on the request.

III J 3. TEXT field syntax

Code:	FIXED(2)
Parameters:	VARIABLE(?)

III J 4. TEXT field semantics

III J 4 a. Code

This field contains an encoding of the type of request. The codes are:

0	Report status
1	Send control function
2	Handle Telnet options and FTP replies
3	Modify suggested allocation

III J 4 b. Parameters

This field contains the parameters for specifying the requests. The contents will vary according to the type of request.

III J 4 b (1). Report status

No parameters are required.

III J 4 b (2). Send control function

III J 4 b (2) (a). Parameter field syntax

Control Function:	FIXED (8)

III J 4 b (2) (b). Parameter field semantics

This field contains a parameter specifying a Telnet control function that the User FTP module should send on the control connection. The parameter values and their meanings are:

242	SYNCH
243	Break
244	Interrupt process
245	Abort output
246	Are you there

III J 4 b (3). Handle options and FTP replies

III J 4 b (3) (a). Parameter field syntax

Option control:	FIXED(2)
Reply text control:	FIXED(1)

III J 4 b (3) (b). Parameter field semantics

These fields contain parameters indicating 1) how the front end should respond to Telnet option negotiations on the control connection 2) whether or not the front end should forward the text of FTP replies to the host. How Telnet options and FTP replies are handled before the process sends the relevant EXECUTE Commands is installation-specific and must be specified in the adaptation description. The parameter values and their meanings are:

Option control values:

1	Refuse all options
2	Refuse no options
3	Refuse options not handled by the front end

Reply text control values:

0	Send FTP reply text
1	Don't send FTP reply text

III J 4 b (4). Modify suggested allocation

III J 4 b (4) (a). Parameter field syntax

Messages:	FIXED(16)
Bits:	FIXED(32)

III J 4 b (4) (b). Parameter field semantics

The semantics of this field are identical to those described for the allocation fields in the BEGIN Command (see sections III A 4 b (4) - III A 4 b (5).)

III K. EXECUTE Response

III K 1. When Sent

The service module sends an EXECUTE Response when

a) it has completed the operation requested by an EXECUTE Command, or
b) it has received an EXECUTE Command whose TEXT is in error.

The process does not send EXECUTE Responses for this process-to-service protocol.

III K 2. Action on Receipt

When the process receives an EXECUTE Response, it should examine the TEXT to see whether the Response indicates an error. If no error is indicated, it may consider that the action requested by a previous EXECUTE Command was successful. If an error is indicated, appropriate error recovery action should be initiated. This action is idiosyncratic to the process.

III K 3. TEXT field syntax

Result:	FIXED(2)
Ref-Code:	FIXED(2)
Connection-info:	COMPLEX(?)
Control connection-state:	FIXED(2)
Data connection-state:	FIXED(2)
Host:	FIXED(32)
Foreign Socket:	FIXED(32)
Messages:	FIXED(16)
Bits:	FIXED(32)
Local Socket:	FIXED(32)

III K 4. TEXT field semantics

III K 4 a. Result

This field contains an encoding of the result of the request (made in a previous EXECUTE Command) to which the EXECUTE Response is a reply. The codes are:

0	Success
1	Illegal code in request
2	TEXT too short.

III K 4 b. Ref-code

This field contains the same value as the Code field of the TEXT of the EXECUTE Command to which the EXECUTE Response is a reply.

III K 4 c. Connection-info

This field is present only if the Ref-code has a value of zero or three (status or modify allocation). The semantics of this field are identical with those of the Connection-info field of the BEGIN Response (section III F 4), with the exception of the Status field, which has been divided into two fields with the following meanings.

III K 4 c (1). Control connection-state

If the ref-code value is zero (status), this field indicates the present state of the FTP control connection. The possible values of this field and their meanings are:

0	Closed
1	Open
2	Waiting for open to complete
3	Waiting for close to complete

III K 4 c (2). Data connection-state

If the ref-code value is zero (status), this field indicates the present state of the FTP Data connection. The possible values of this field and their meanings are:

0	Closed
1	Open
2	Waiting for open to complete
3	Waiting for close to complete

IV. Details of Offloading Server FTP

The ARPANET File Transfer Protocol (FTP) consists of two major sections: a command-and-response section and a data transfer section. The file access process-to-service protocol deals with offloading the data transfer section and the FTP commands for data transfer (RETRIEVE, STORE, and APPEND). The Server FTP process-to-service protocol deals with offloading the command-and-response section. We here describe some of the aspects of offloading the Server FTP. These aspects include session establishment and control, user authentication, restart, and interaction with the file access module.

Session establishment and control. There are several possible ways in which service could be initiated and controlled. The scheme described in this protocol appears to be the simplest and most flexible. In this scheme, BEGIN Commands only go from the host to the front end. When the front end service module receives a BEGIN Command, it sets up a connection as requested. In most cases, the service module will not be requested to actively initiate a connection but only to set up a listen on socket 3 and wait for a request for service from a remote host. After a connection has been established, the service module sends a BEGIN Response to the host. The host process must generate a distinct BEGIN Command for each connection it is willing to accept. Thus, the host is able to limit the number of

FTP users by withholding BEGIN Commands. The front end may also limit demand on its resources by refusing BEGIN Commands from the host.

If user authentication is done in the front end, then an alternative method may be used. Since most FTP sessions consist of logging in, moving one or more files, and then logging out, virtually no traffic is sent on the Server FTP channel. Everything is handled by the server FTP service module (SFSM) and the file access modules, unless the user does a DELETE or RENAME. Therefore, it makes sense to leave control of the service in the front end. The SFSM in the front end would listen on socket 3 as usual. When a user connected to the FTP contact socket, a connection would be established. The user would be logged in, and data transfer parameters set. When the user sent a data transfer command, the UFAM would be asked to set up a session. At this point the host could refuse access if it were too heavily loaded. Normally the transfer and the session would be done without requiring the use of a Server FTP PSP channel.

<u>User authentication and security</u>. The ARPANET FTP provides three commands for authenticating users: USER, PASS, and ACCT. Depending on the security arrangement between the host and the front end, it may be possible to offload these commands. This, however, implies that the host trusts the front end's security as much as it trusts

it own. Some systems may wish to restrict network access to data transfers. For example, a data base system might allow remote network users to query and even to update data bases and to transfer files in and out, but the system might wish to restrict delete and rename capabilities to local users only. In this case, the Server FTP PSP would not be used at all, and the Server FTP module in the front end would use the File Access PSP only.

<u>Restart</u>. An important aspect of FTP that can be offloaded is the facility to restart a transfer that has been aborted because of a host or network failure. There are two cases to consider: data going out to the net and data coming in from the net. Let us first consider data moving out. If the front end can exert enough addressing control through the FA-PSP, it is possible to allow the FAS in the front end to insert the restart markers, thus offloading this function from the host. The only requirement is that the FAS in the front end be able to address the same point in the file when given the restart marker and commence the transfer at that point. If this is not possible then these functions must be in the host.

For data coming into the host, it is necessary for the Server FTP to be able to report to the User FTP when the data associated with a particular marker has been entered into the file, and to associate with the user-specified markers a locally generated marker that can be used to

restart the transfer at this point.

Interaction with the file access module. Offloading Server FTP will in most cases require the Using File Access Module (UFAM) to be in the front end. The Server FTP Service Module (SFSM) acts as an intelligent message switch. The SFSM does not actually perform any of the FTP commands, except possible those having to do with connection control (e.g., BYE or REINITIALIZE) or user authentication. When the SFSM receives a command that is not offloaded, it will pass the command on to the residual Server FTP in the host. (The command may be first translated into some other form. If the command is one that is offloaded (e.g., one of the data transfer parameter commands), the SFSM will use its knowledge of the facilities supported in the front end and send a suitable reply to the foreign host. This information is then communicated to the UFAM to specify the translation requirements for the transfer. The SFSM may hold parameter information until a data transfer command is received and then relay all information specifying the transfer (including the data connection sockets) or the parameters may be relayed as they are determined depending on the implementations and the IPC facilities in the front end.

If the translation is done in the front end, the UFAM in the front end requires that only the file parameters be specified in the BEGIN Command. If, however, data

translation is done in the host, the data transfer parameters must also be present. (See [2].)

<u>Content of TRANSMIT Commands</u>. In this protocol, the TRANSMIT Command is used to send FTP commands and replies between the service module and the host. Some installations may wish to send some interactive form or encoding peculiar to their host's file system and let the service module in the front end translate it into the proper FTP commands. For most installations, the best method of encoding this information is to just send the FTP commands themselves. Since many of the parameters are character strings, little is saved by additional encodings. The only major exceptions would be the data transfer parameter commands such as TYPE, MODE, etc., which are not sent to the host in most Server FTP offloading schemes. Similar arguments hold for the replies. Because of the semantics attached to the individual decimal digits, some encoding is possible, but it is of limited utility. Some installations may wish to use the control function described in section III J 4 b (3) to suppress the text associated with replies. Server FTP offloadings that also support file systems in the front end may find it useful to pass only reply codes from the host and generate a consistent set of reply text from the front end. For a suggested encoding for FTP commands and replies, see Section V of [1].

V. Scenarios

User authentication in the host. Consider the following scenario for an offloaded Server FTP. We will assume that user authentication and the FTP commands USER, PASS, and ACCT are done in the host. We will also assume that the seven FTP commands listed in Section II are done in the host. All other FTP commands are to be handled by the front end. Further, the data transfer process is also in the front end and the file access module is used to transfer the file from the host. We will not describe the behavior of the file access module in detail. For a scenario describing its use see the specification of the File Access Process-to-Service Protocol [2]. Also we will not be concerned with the detailed functioning of the channel protocol.

First, the residual Server FTP module (SFM) in the host will send a BEGIN Command to the front end offering FTP Service:

<BEGIN><C>

Since no parameters are present, the defaults are assumed. The Server FTP Service Module (SFSM) will listen for an ICP on socket 3 (the FTP contact socket) from any host for an indefinite period of time.

Some time later, a remote user connects to socket 3.

The ICP protocol is performed and the Telnet control connection is opened. The SFSM will then send the FTP herald message (a 300 reply) to the remote host and a BEGIN Response to the local host.

<BEGIN><R><success><Host=12><Foreign Socket=57834>
<Messages=4><Bits=2000><Local Socket=2478>

In order to support another user, the host will now send another

<BEGIN><C>.

When the FTP USER command is received from the remote user, the SFSM sends it to the SFM:

<TRANSMIT><C>USER day

(Since TRANSMIT Responses do not require any action by the services or the process they are not shown here.)

The SFM replies with

<TRANSMIT><C>331 Please send pass command.

The SFSM sends the TEXT of this command over the net to the user. The user responds by sending the FTP PASS command, which is then relayed to the SFM:

<TRANSMIT><C>PASS john

The SFM replies with

<TRANSMIT><C>230 logged in. please proceed.

The user then sends an FTP RETR command requesting that a file "junk.note" be moved from the local host to the foreign (i.e., the user's) host. The SFSM will then

1) do a listen on the default data socket, and
2) pass the command, the user identification parameters, and information on the local socket or port to the Using File Access Module (UFAM) in the front end.

(Since no data transfer parameter commands were received, the defaults are assumed.

The UFAM will then establish a session with the Serving FAM (SFAM) in the host and attempt to access the file. When a successful BEGIN Response is returned, the UFAM will open the data connection and notify the SFSM that the transfer can begin. The SFSM will now send the FTP 150 reply to the remote host indicating that the transfer is ready to start. The UFAM will then transfer the data to the network.. When the transfer is complete, the UFAM will notify the SFSM which will send the FTP 250 reply to the remote host indicating the end of transfer.

Now the user decides to delete the file he has just transferred, so he sends an FTP DELE command to the SFSM. the SFSM recognizes that this is not a command it does and passes it on to the host:

<TRANSMIT><C>DELE junk.note

The host deletes the file and returns:

<TRANSMIT><C>250 File junk.note deleted.

Finally the user is finished, and sends the FTP BYE command. The SFSM will send an FTP 221 reply indicating that it is closing the Telnet connection, close the data connection, notify the UFAM to terminate its session, and terminate its own session with the host by sending an END Command:

<END><C><3> (indicating a user-requested close)

The host will respond with

<END><R>

and the interaction is complete.

User authentication in the front end. Let us consider how this protocol would be used if user authentication is done in the front end. Let us assume that the only FTP commands handled by the host are the seven listed in Section II. We will also assume that the data transfer commands are handled by the Using File Access Module (UFAM) in the front end.

The Server FTP Service Module (SFSM) is offering FTP service by listening on socket 3 (the FTP contact socket). A user at some remote host does an ICP to the contact socket and a control connection is established. The user

sends the FTP USER and PASS commands and is logged in by the SFSM for file transfer service. The user then sends a RETR command to transfer the file "junk.note" to his host. The necessary information, including pathname, user identification, and network data sockets, are passed to the UFAM. The UFAM establishes a session and transfers the file. After the transfer is complete, the user wishes to delete the same file and sends the FTP command:

DELE junk.note

This requires establishing a Server FTP PSP channel to the host, since one has not been required as yet. The SFSM sends a BEGIN Command to the front end:

<BEGIN><C><security=day,john>

The Server FTP Module (SFM) in the host acknowledges with:

<BEGIN><R>

The SFSM then sends over the command:

<TRANSMIT><C>DELE junk.note

(Since TRANSMIT Responses do not require any action by either the process or the service, they are not shown in this example.) The file is deleted and the reply sent back to the SFSM:

<TRANSMIT><C>250 File junk.note deleted.

The TEXT of this TRANSMIT Command is then sent over the control connection to the user. The user decides he is finished and sends an FTP BYE Command to the front end. The SFSM immediately sends the proper FTP reply to the user, notifies the UFAM to terminate its session and sends an END Command to the SFM in the host:

<END><C><0> (indicating normal completion)

The SFM replies immediately with:

<END><R>.

Note that if the user had not asked to delete the file this PSP would not have been used at all.

VI. References

1. Day, J. "User FTP Process-to-Service Protocol," CAC Technical Memorandum No. 101, Center for Advanced Computation, University of Illinois at Urbana-Champaign, 1977.

2. Day, J. "File Access Process-to-Service Protocol," CAC Technical Memorandum No. 102, Center for Advanced Computation, University of Illinois at Urbana-Champaign, 1977.

3. Day, J. "Offloading ARPANET Protocols to a Front End," CAC Document No. 230, Center for Advanced Computation, University of Illinois at Urbana-Champaign, 1977.

4. Neigus, N. "The File Transfer Protocol," ARPANET RFC 542, 1973.

5. Neigus, N.; Pogran, K.; and Postel, J. "A New Schema for FTP Reply Codes," ARPANET RFC 640, 1974.

6. Postel, J. "Telnet Protocol Specification," NIC #18639, 1973.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER CAC Document Number 230 CCTC-WAD Document Number 7511	2. GOVT ACCESSION NO.
3. RECIPIENT'S CATALOG NUMBER	
4. TITLE (and Subtitle) Network Research in Front Ending and Intelligent Terminals - Offloading ARPANET Protocols to a Front End	5. TYPE OF REPORT & PERIOD COVERED Research
	6. PERFORMING ORG. REPORT NUMBER CAC #230
7. AUTHOR(s) John D. Day	8. CONTRACT OR GRANT NUMBER(s) DCA100-76-C-0088
9. PERFORMING ORGANIZATION NAME AND ADDRESS Center for Advanced Computation University of Illinois at Urbana-Champaign Urbana, Illinois 61801	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Command and Control Technical Center WWMCCS ADP Directorate, 11440 Isaac Newton Sq., N. Reston, Virginia 22090	12. REPORT DATE September 30, 1977
	13. NUMBER OF PAGES 262
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)	15. SECURITY CLASS. (of this report) UNCLASSIFIED
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Copies may be requested from the address given in (11) above.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

No restriction on distribution.

18. SUPPLEMENTARY NOTES

None

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Network front end

Network protocols

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The CAC is engaged in an investigation of the benefits to be gained by employing a network front end. This document presents a discussion of alternative strategies for offloading ARPANET protocol implementations to a front end. A set of six appendices specify the host-to-front-end, process-to-service protocols required to implement the various strategies.